Oracle® Retail Back Office

Operations Guide

Release 13.0.3

September 2009

ORACLE®

Oracle Retail Back Office Operations Guide, Release 13.0.3

Primary Author: Graham Fredrickson

Value-Added Reseller (VAR) Language

Oracle Retail VAR Applications

The following restrictions and provisions only apply to the programs referred to in this section and licensed to you. You acknowledge that the programs may contain third party software (VAR applications) licensed to Oracle. Depending upon your product and its version number, the VAR applications may include:

(i) the software component known as **ACUMATE** developed and licensed by Lucent Technologies Inc. of Murray Hill, New Jersey, to Oracle and imbedded in the Oracle Retail Predictive Application Server - Enterprise Engine, Oracle Retail Category Management, Oracle Retail Item Planning, Oracle Retail Merchandise Financial Planning, Oracle Retail Advanced Inventory Planning, Oracle Retail Demand Forecasting, Oracle Retail Regular Price Optimization, Oracle Retail Size Profile Optimization, Oracle Retail Replenishment Optimization applications.

(ii) the **MicroStrategy** Components developed and licensed by MicroStrategy Services Corporation (MicroStrategy) of McLean, Virginia to Oracle and imbedded in the MicroStrategy for Oracle Retail Data Warehouse and MicroStrategy for Oracle Retail Planning & Optimization applications.

(iii) the **SeeBeyond** component developed and licensed by Sun MicroSystems, Inc. (Sun) of Santa Clara, California, to Oracle and imbedded in the Oracle Retail Integration Bus application.

(iv) the **Wavelink** component developed and licensed by Wavelink Corporation (Wavelink) of Kirkland, Washington, to Oracle and imbedded in Oracle Retail Mobile Store Inventory Management.

(v) the software component known as **Crystal Enterprise Professional and/or Crystal Reports Professional** licensed by SAP and imbedded in Oracle Retail Store Inventory Management.

(vi) the software component known as **Access Via™** licensed by Access Via of Seattle, Washington, and imbedded in Oracle Retail Signs and Oracle Retail Labels and Tags.

(vii) the software component known as **Adobe Flex™** licensed by Adobe Systems Incorporated of San Jose, California, and imbedded in Oracle Retail Promotion Planning & Optimization application.

(viii) the software component known as **Style Report™** developed and licensed by InetSoft Technology Corp. of Piscataway, New Jersey, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.

(ix) the software component known as **DataBeacon™** developed and licensed by Cognos Incorporated of Ottawa, Ontario, Canada, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.

You acknowledge and confirm that Oracle grants you use of only the object code of the VAR Applications. Oracle will not deliver source code to the VAR Applications to you. Notwithstanding any other term or condition of the agreement and this ordering document, you shall not cause or permit alteration of any VAR Applications. For purposes of this section, "alteration" refers to all alterations, translations, upgrades, enhancements, customizations or modifications of all or any portion of the VAR Applications including all reconfigurations, reassembly or reverse assembly, re-engineering or reverse engineering and recompilations or reverse compilations of the VAR Applications or any derivatives of the VAR Applications. You acknowledge that it shall be a breach of the agreement to utilize the relationship, and/or confidential information of the VAR Applications for purposes of competitive discovery.

The VAR Applications contain trade secrets of Oracle and Oracle's licensors and Customer shall not attempt, cause, or permit the alteration, decompilation, reverse engineering, disassembly or other reduction of the VAR Applications to a human perceivable form. Oracle reserves the right to replace, with functional equivalent software, any of the VAR Applications in future releases of the applicable program.

Contents

9 Commerce Services

List of Figures

List of Examples

List of Tables

Preface

Oracle Retail Operations Guides contain the requirements and procedures that are necessary for the retailer to configure Back Office, and extend code for a Back Office implementation.

Audience

This document is intended for Oracle Retail Back Office administrators and developers who develop code for a Back Office implementation.

Related Documents

For more information, see the following documents in the Oracle Retail Back Office Release 13.0.3 documentation set:

- Oracle Retail Strategic Store Solutions Licensing Information
- Oracle Retail Back Office documentation set
- Oracle Retail Labels and Tags documentation set
- Oracle Retail Central Office documentation set
- Oracle Retail Point-of-Service documentation set
- Oracle Retail Mobile Point-of-Service documentation set
- Oracle Retail Strategic Store Solutions Implementation Guide

Customer Support

To contact Oracle Customer Support, access My Oracle Support at the following URL:

- `https://metalink.oracle.com`

When contacting Customer Support, please provide the following:

- Product version and program/module name
- Functional and technical description of the problem (include business impact)
- Detailed step-by-step instructions to recreate
- Exact error message received
- Screen shots of each step you take

Review Patch Documentation

If you are installing the application for the first time, you install either a base release (for example, 13.0) or a later patch release (for example, 13.0.3). If you are installing a software version other than the base release, be sure to read the documentation for each patch release (since the base release) before you begin installation. Patch documentation can contain critical information related to the base release and code changes that have been made since the base release.

Oracle Retail Documentation on the Oracle Technology Network

In addition to being packaged with each product release (on the base or patch level), all Oracle Retail documentation is available on the following Web site (with the exception of the Data Model which is only available with the release packaged code):

`http://www.oracle.com/technology/documentation/oracle_retail.html`

Documentation should be available on this Web site within a month after a product release. Note that documentation is always available with the packaged code on the release date.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Backend System Administration and Configuration

This chapter discusses the configuration steps that must be performed after Oracle Retail Back Office is deployed.

- Loading Oracle Retail Labels and Tags
- Security Configuration
- Password Policy
- Starting Up the Application
- Importing Parameters
- Scheduling Post-processors

Note: Before making changes to default settings in the Back Office application, make sure the changes will not affect the PABP settings.

For more information on PABP, see the *Oracle Retail Strategic Store Solutions Security Implementation Guide*. The guide is available on Metalink:

Metalink Note: 567438.1

Loading Oracle Retail Labels and Tags

For loading Labels and Tags, Oracle Retail provides an SQL script, `ant init_labels`. This script is located in the `backofficeDBInstall.jar` file.

Security Configuration

You can use your own security system or you can use the security provided with the Oracle Retail Back Office database. The database requires certain security information, which can either come from another database or be set manually.

Oracle Retail Back Office offers more than 70 security access points that control access to different functions. Work with Oracle Retail to map your business security roles to the roles provided.

Oracle Retail Back Office also enables you to control workflow through approval permissions, which require that certain tasks scheduled by employees must be approved by other employees before they can take place.

Security roles are listed in the `\applications\backoffice\deploy\backoffice.ear\application.xml` file.

An additional consideration is browser security. When multiple users use the same systems, savvy end-users could use the browser history trail to try to access data not appropriate for their access levels. Configure the browsers used to access the application to rdisable the history function and the address bar to prevent this.

Security Implementation — Warnings and Advice

Oracle Retail is committed to providing our customers software that, when combined with overall system security, is capable of meeting or exceeding industry standards for securing sensitive data. By maintaining solutions based on standards, Oracle Retail provides the flexibility for retailers to choose the level and implementation of security without being tied to any specific solution.

Each retailer should carefully review the standards that apply to them with special emphasis on the Payment Card Industry (PCI) best practices. The Oracle Retail applications represent one portion of the entire system that must be secured; therefore, it is important to evaluate the entire system including operating system, network, and physical access.

The following recommendations are required by Visa:

1. Don't use database or operating systems administrative accounts for application accounts. Administrative accounts and any account that has access to sensitive data should require complex passwords as described below. Always disable default accounts before use in production.
2. Assign a unique account to each user. Never allow users to share accounts. Users that have access to more than one customer record should use complex passwords.
3. Complex passwords should have a minimum length of 7 characters, contain both numeric and alphabetic characters, be changed at least every 90 days, and not repeat for at least 4 cycles.
4. Unused accounts should be disabled. Accounts should be temporarily disabled for at least 15 minutes after six invalid authentication attempts.
5. If sensitive data is transmitted over a wireless network, the network must be adequately secure, usually through use of WPA, 802.11i, or VPN.
6. Never store sensitive data on machines connected to the internet. Always limit access using a DMZ and/or firewall.
7. For remote support, be sure to use secure access methods such as two-factor authentication, SSH, SFTP, and so forth. Use the security settings provided by third-party remote access products.
8. When transmitting sensitive data, always use network encryption such as SSL.

Following these recommendations does not necessarily ensure a secure implementation of the Oracle Retail products. Oracle recommends a periodic security audit by a third-party. Review the PCI standards for additional information.

Password Policy

One of the most efficient ways to manage user access to a system is through the use of a password policy. The policy can be defined in the database. One policy is defined and applied to all users for Oracle Retail Back Office. The Password Policy consists of the following set of out-of-the-box criteria. For this release, customizing the password policy criteria is permitted through enabling status code system settings and updating password policy system settings to the desired setting.

In order to be PCI compliant, the Password Policy needs to be set to the following:

- Force user to change password after 90 days.
- Warn user of password expiration 5 days before password expires.
- Lock out user 3 days after password expires or password is reset.
- Lock out user after 6 consecutive invalid login attempts.
- Password must be at least 7 characters in length.
- Password must not exceed 22 characters in length.
- Password must not match any of the 4 previous passwords.
- Password must include at least 1 alphabetic character.
- Password must include at least 1 numeric character.

Once the desired password policy has been defined, it is applied to all authorized users of the Oracle Retail Point-of-Service, Oracle Retail Mobile Point-of-Service, Oracle Retail Back Office, Oracle Retail Labels and Tags, and Oracle Retail Central Office applications. The password policy must be defined once per database.

Note: Default settings for Password Policy are PCI-DSS compliant. Changes to these default settings can invalidate this compliance.

Password Reset

Users locked out of the system must request the assistance of an administrator to have their password reset. The administrator resets the password by selecting the reset password option in Back Office when applicable. When a user password is reset the system generates a temporary random password. The reset password status is immediately set to "expired", prompting the user to change the temporary password at the next successful login.

Each time a password is changed, the previous password is stored subject to the "Passwords must not match any of the N previous passwords" criteria set for the policy associated with the assigned user role. Temporary passwords might not comply with the password policy and are not stored in the password list.

Do the following to change the password of another user:

1. Log in.
2. Click **Employee**.
3. Click **Search** in the left navigation panel.
4. Search for the user whose password you are resetting. You can search by user ID or name. Click **Search**.
5. Click on the user ID in the Employee Select screen. This opens the Employee Master screen.

6. Make sure User Info and Role Assignments information is accurate.
7. Click **Reset Password**.

   You will see a message asking if you are sure you want to reset the password. Click **Yes**.
8. A screen with the user's new temporary password is shown.

   Note: This temporary password is provided on this screen only. Record this temporary password. The password is not recorded or logged, and is not provided by email. Administrators must provide this temporary password to the user.

9. Click **Enter**.

Password Change

Do the following to change your password:

1. Log in.
2. Click **Change Password** in the left navigation bar.
3. Enter your current password.
4. Enter a new password.
5. Enter the new password again.
6. Click **Update**.
7. You will see a screen with the following message:

   ```
   Your password has been changed.
   Use this password the next time you log in.
   ```
8. Click **Enter**.

Adding User

Do the following to add a user:

1. Log in.
2. Click **Employee**.
3. Click **Add**.
4. Enter the following:
   - First name
   - Last name
   - User ID
5. Provide a Role, for example, Administrator.
6. Select a status, for example, Active.
7. Click **Save**.
8. A screen with the new user's temporary password is shown.

Note: This temporary password is provided on this screen only. Record this temporary password. The password is not recorded or logged, and is not provided by email. Administrators must provide this temporary password to the user.

Starting Up the Application

To perform some final configuration tasks, such as importing parameters, requires running the application.

To run Oracle Retail Back Office:

1. Verify that the application is available in the Application Server environment.
2. Access the application from a browser, using the following URL format:

   `https://<appserver-hostname>:<application port>/backoffice`

Importing Parameters

Note: An initial set of parameters must be imported before you can use Oracle Retail Back Office.

This section provides an overview of the procedures for importing an initial set of parameters. The procedure for importing parameters through the application user interface are described in more detail in the *Oracle Retail Back Office User Guide*.

For information on specific parameters, see the Oracle Retail Strategic Store Solutions Configuration Guide.

Importing Initial Parameters

To import the initial parameters through the user interface:

1. Click the **Data Management** tab. The Available Imports screen is displayed.
2. To import the master parameter set, click the **File** link in the Import Parameters row. Click **Browse** to import `parameterset.xml` from the *`<Back Office install directory>`*`/backoffice/db` folder.
3. To import the initial set of Oracle Retail Back Office application parameters, click the **File** link in the Import Application Parameters row. Click **Browse** to import `backoffice.xml` from the *`<Back Office install directory>`*`/backoffice/db` folder.

To import parameters using an ant target:

1. Change to the *`<Back Office install directory>`*`/backoffice/db` directory.
2. Edit the `db.properties` file. Ensure that the properties that affect parameter loading are properly set.
3. Execute the following command:

   ```
   ant load_parameters
   ```

Note: Make sure that the Apache Ant utility is in your PATH. You can find it in `thirdparty/apache-ant-1.6.2/bin`.

Scheduling Post-processors

Schedule post-processor jobs after installing Oracle Retail Back Office.

To schedule regular post-processor jobs within Oracle Retail Back Office:

1. Select **Admin** and then **Job Manager**.
2. From the list of import options, select **Available Imports**.
3. From the available imports, click **Schedule** adjacent to the Transaction Post Processor. The Job Schedule page is displayed.
4. Choose **Scheduled**. Additional scheduling options are displayed.
5. Enter the current date in the **Begin Date** field.
6. Check the **Repeating** check box.
7. Leave the **No End** radio button selected.
8. Set the **Repeating** options **Daily** and **Interval**.
9. Enter a run time in the appropriate box.
10. Click **Add**. The time you entered is displayed in the Scheduled Times section at the bottom of the screen.
11. Click **Next**. The Notification screen is displayed.
12. Add the e-mail addresses of anyone you want to be notified about the post-processor job.
13. Click **Next**. The Distribution Summary screen opens, with a summary of the post-processor job displayed in the Task Information box.
14. Click **Submit Job**. The Distribution Confirmation screen opens.
15. Click **Done**.

Modifying Help Files

Back Office online help is created using Oracle Online Help for the Web. Information on this technology is available at
`https://www.oracle.com/technology/docs/tech/java/help/index.html`

The online help is generated from the Back Office User Guide. Each chapter in the user guide is divided into sections. You can look at the Table of Contents for the user guide to see how each chapter is structured. When the user guide is converted into online help, each section is converted into an html help file.

Some help files contain specific information for a screen. Other help files have the background or topic information that is contained in the user guide. For screen help, the name of the file includes the name of the screen. For background help, the name of the file is based on the section in the user guide. For example, the help file for the Employee Master screen is named employeemasterhelp.htm. The information in the Job Manager section is in the jobmanagerhelp.htm file.

The backoffice.ear file contains the backoffice-help.war. The war file contains the following:

```
helpsets folder
    bo_olh folder
        dcommon folder (definitions for styles, gif files for buttons)
        img folder (any images included in the online help from the user guide)
        help files
```

Note: If you have Labels and Tags installed, online help is in `lt_olh`. You will have both `bo_olh` and `lt_olh`.

To update a help file:

1. Locate the help file to be changed.
2. Edit the help file.
3. Replace the updated file in the helpset and in backoffice.ear.
4. Redeploy backoffice.ear.

2 Technical Architecture

This chapter describes the main layers of the application, and goes into some detail about the middle tier's use of a model-view-controller (MVC) pattern. The remainder of this overview covers the top-level tier organization of the application and how the application relates to other Oracle Retail applications in an enterprise environment. This guide assumes a basic familiarity with the J2EE specification and industry standard software design patterns.

The architecture of Back Office reflects its overriding design goals:

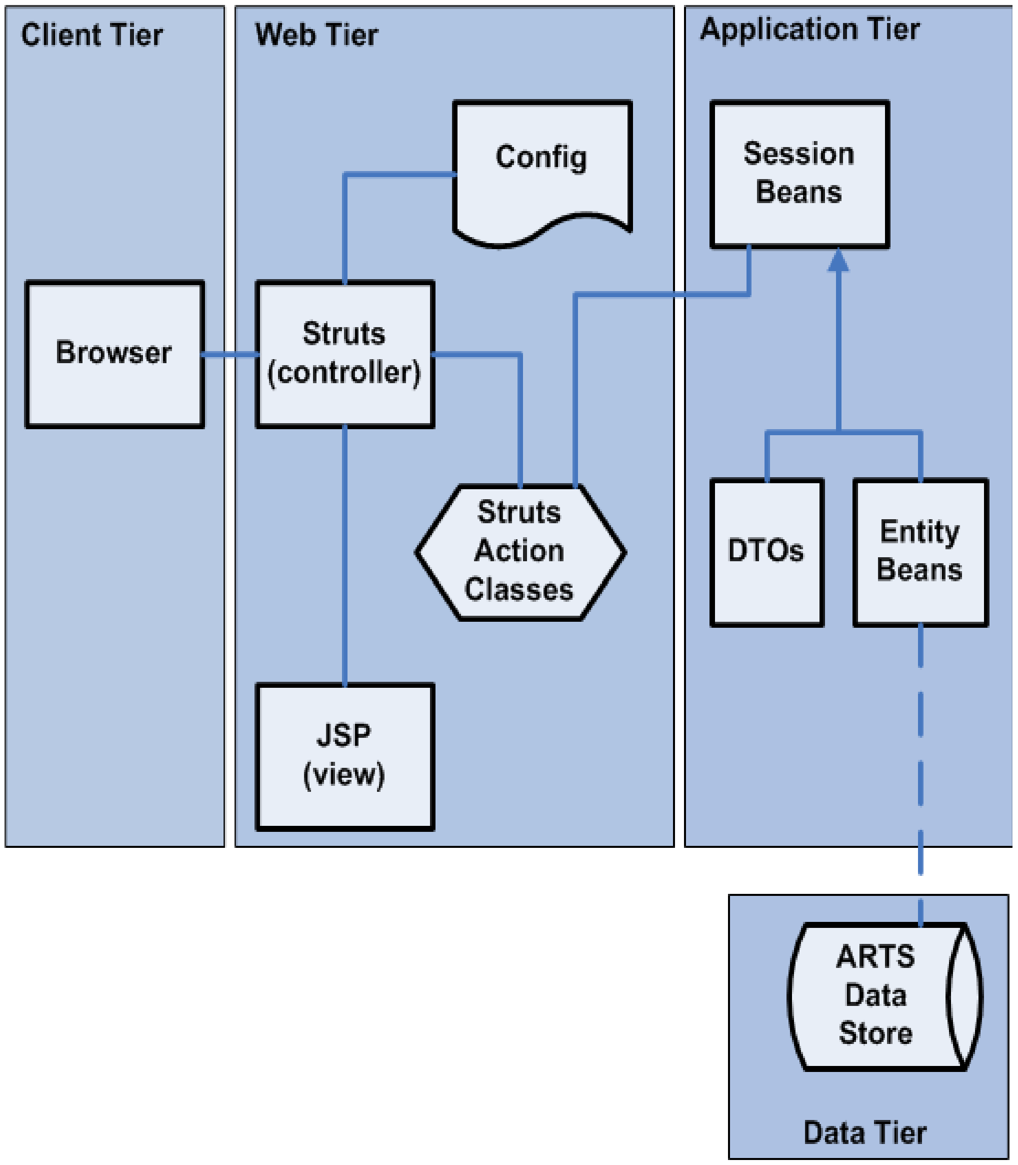
- Well-defined and decoupled layers
- Use of appropriate J2EE standards
- Leveraging other open standards where possible

Tier Organization

The architecture of Back Office uses client, middle, and data tiers. The client tier is a Web browser; the middle tier is deployed on an application server; and the data tier is a database deployed by the retailer.

The middle tier is organized in an MVC design pattern, also called a Model 2 pattern. This chapter focuses on the middle tier and the model, view, and controller layers that it is divided into.

Figure 2–1 High-Level Architecture



Client Tier

The client system uses a Web browser to display data and a GUI generated by the application. Any browser which supports JavaScript, DHTML, CSS, and cookies can be used. In practice, only a few popular browsers are tested.

Middle Tier

The middle tier of the application resides in a J2EE application server framework on a server machine. The middle tier implements the MVC pattern to separate data structure, data display, and user input.

Model

The model in an MVC pattern is responsible for storing the state of data and responding to requests to change that state which come from the controller. In Back Office this is handled by a set of Commerce Services, which encapsulates all of the business logic of the application. The Commerce Services talk to the database through a persistence layer of entity EJBs, using bean-managed persistence.

Commerce Services are components that have as their primary interface one or more session beans, possibly exposed as Web services, which contain the shared retail business logic. Commerce Services aggregate database tables into objects, combining sets of data into logical groupings. Commerce Services are organized by business logic categories rather than application functionality. These are services like Transaction, Store Hierarchy, or Parameter that would be usable in any retail-centric application.

These services in turn make use of a persistence layer made up of entity beans. Each Commerce Service talks to one or more entity beans, which map the ARTS standard database schema. Using the bean-managed persistence (BMP) pattern, each entity bean maps to a specific table in the schema, and knows how to read from and write to that table. The Commerce Services thus insulate the rest of the application from changes to the database tables. Database changes can be handled through changes to a few entity beans.

The Commerce Services architecture is designed to facilitate changes without changing the product code. For example:

- You can replace a specific component's implementation. For example, the current Store Hierarchy service persists store hierarchy information to the ARTS database. If a customer site has that information in an LDAP server, the Store Hierarchy could be replaced with one that connected to the LDAP. The interface to the service need not change.
- You can create a new service that wraps an existing service (keeping the interface and source code unchanged), but adds new fields. You might create My Customer Service, which uses the existing Customer Service for most of its information, but adds some specific data. All that you change is the links between the Application Manager and the Customer Service. For more information, see Chapter 5, "Commerce Services."

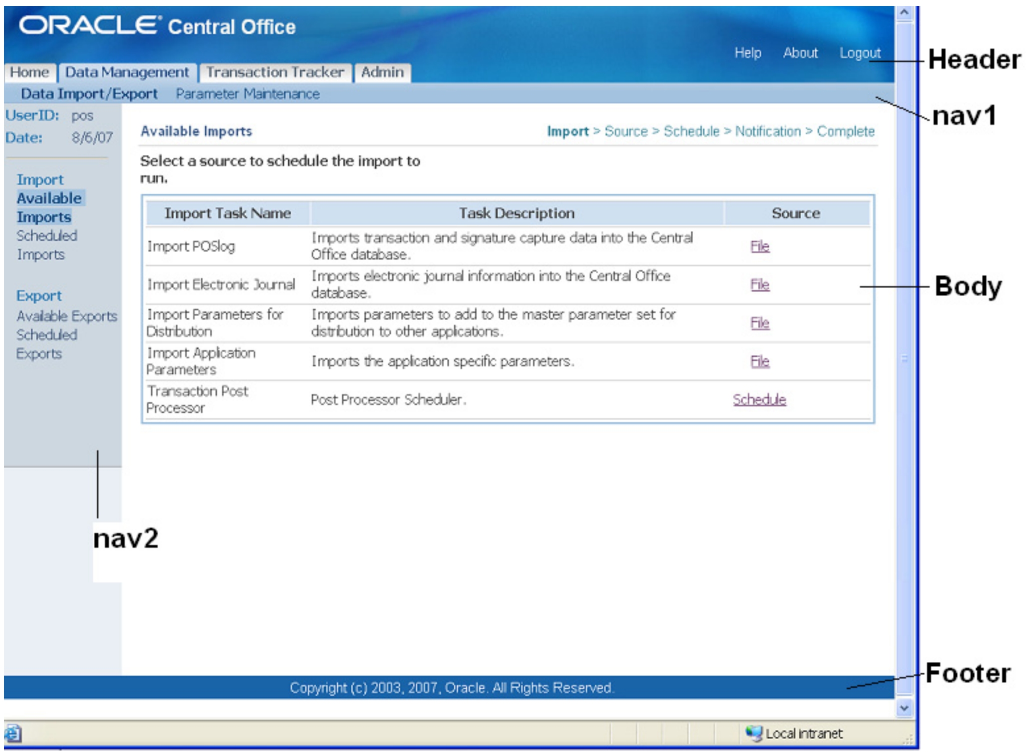
View

The view portion of the MVC pattern displays information to the user. In Back Office this is performed by a Web user interface organized using the Struts/Tiles framework from the open-source Apache Foundation. Using Tiles for page layout enables greater use of the user interface components to enhance the extensibility and customization of the user interface.

To make the view aware of its place in the application, the Struts Actions call into the Application Manager layer for all data updates, business logic, and data requests. Any code in the Struts Actions should be limited to formatting data for the Java server pages (JSPs) and organizing data for calls into the Application Manager layer.

JSPs deliver dynamic HTML content by combining HTML with Java language constructs defined through special tags. Back Office pages are divided into Tiles which provide navigation and page layout consistency.

Figure 2–2 Tiles in an Oracle Retail Application



Select a source to schedule the import to run.

Import Task Name	Task Description	Source
Import POSlog	Imports transaction and signature capture data into the Central Office database.	File
Import Electronic Journal	Imports electronic journal information into the Central Office database.	File
Import Parameters for Distribution	Imports parameters to add to the master parameter set for distribution to other applications.	File
Import Application Parameters	Imports the application specific parameters.	File
Transaction Post Processor	Post Processor Scheduler.	Schedule

Controller

The controller layer accepts user input and translates that input into calls to change data in the model layer, or change the display in the view layer. Controller functions are handled by Struts configuration files and Application Services.

Struts Configuration

The application determines which modules to call, on an action request, based on the struts-config.xml file.There are several advantages to this approach:

- The entire logical flow of the application is in a hierarchical text (xml) file. This makes it easier to view and understand, especially with large applications.
- The page designer does not need to read Java code to understand the flow of the application.
- The Java developer does not need to recompile code when making flow changes.

Struts reads the struts-config.xml once, at startup, and creates a mapping database (a listing of the relationships between objects) that is stored in memory to speed up performance.

Application Services

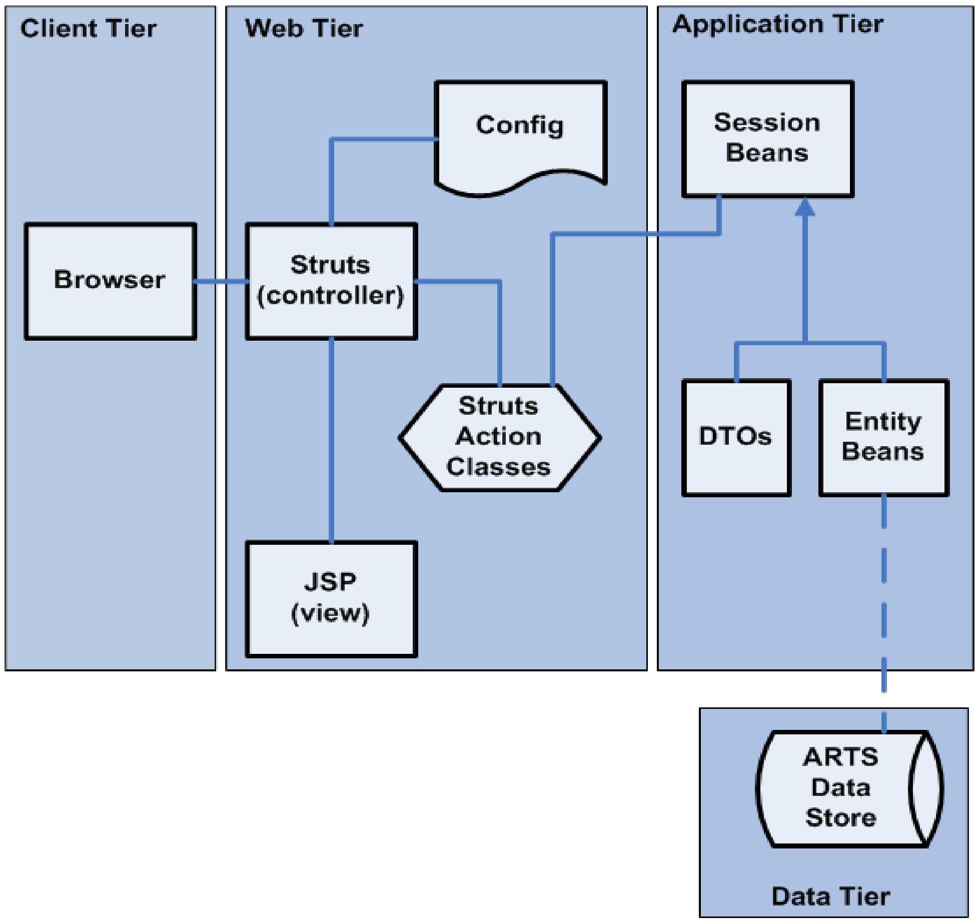
The application services layer contains logical groupings of related functionality specific to the Back Office application components, such as Store Operations. Each grouping is called an application manager. These managers contain primarily application logic. Retail domain logic should be kept out of these managers and instead shared from the Commerce Services tier.

The application services use the Session Facade pattern; each Manager is a facade for one or more Commerce Services. A typical method in the Application Services layer aggregates several method calls from the Commerce Services layer, enabling the individual Commerce Services to remain decoupled from each other. This also strengthens the Web user interface tier and keeps the transaction and network overhead to a minimum.

For example, the logic for assembling and rendering a retail transaction into various output formats are handled by separate Commerce Services functions. However, the task of creating a PDF file is modeled in the EJournal Manager, which aggregates those separate Commerce Service functions into a single user transaction, thus decreasing network traffic and lowering maintenance costs.

For more information, see Chapter 8, "Application Services".

Figure 2–3 Application Manager as Facade for Commerce Services



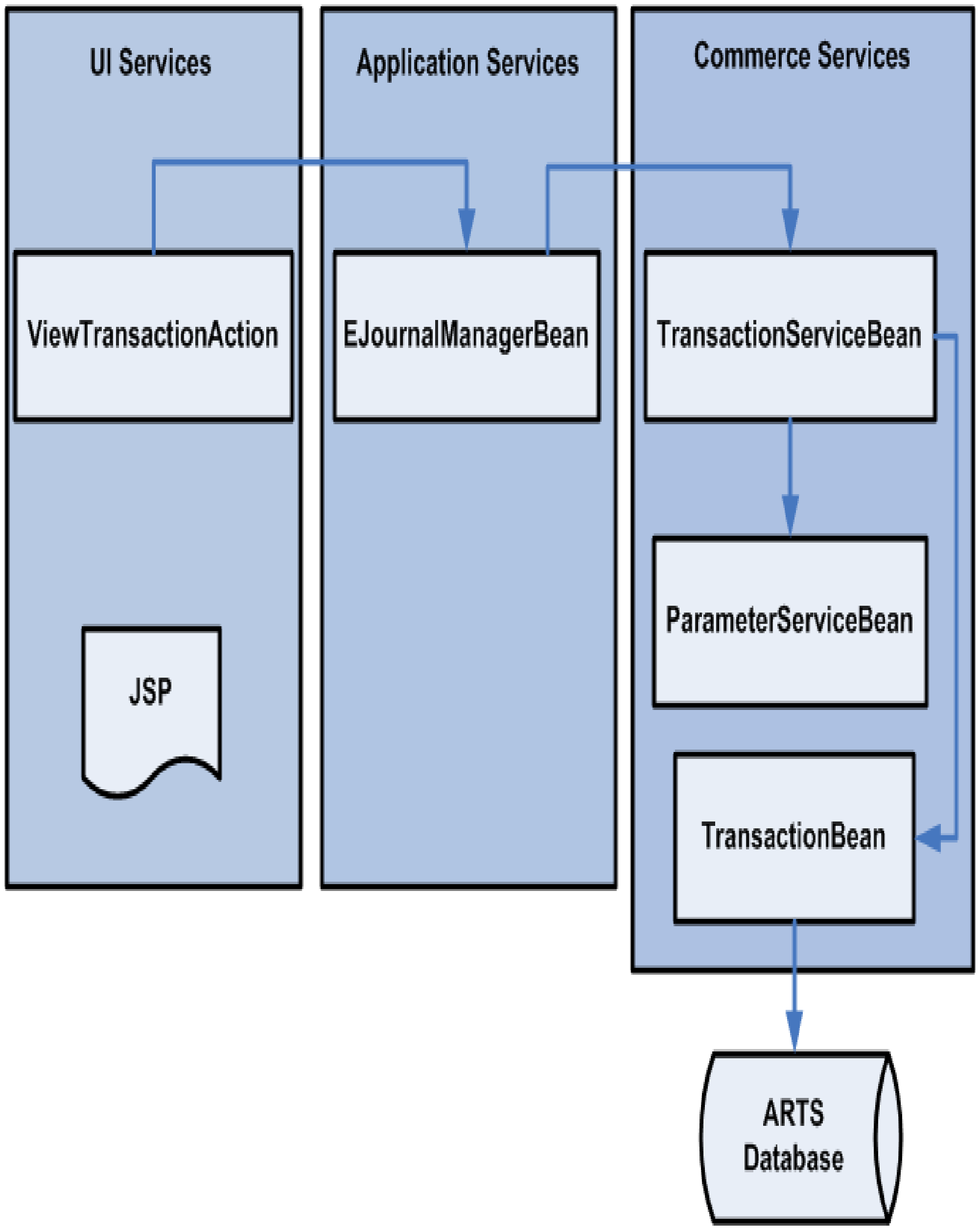
Data Tier

The Data Tier is represented by a database organized using the ARTS standard schema. Customer requirements determine the specific database selected for a deployment. For more information, see Chapter 9, "Commerce Services".

Dependencies in Application and Commerce Services

The following diagram shows representative components Application Services and Commerce Services. Arrows show the dependencies among various components.

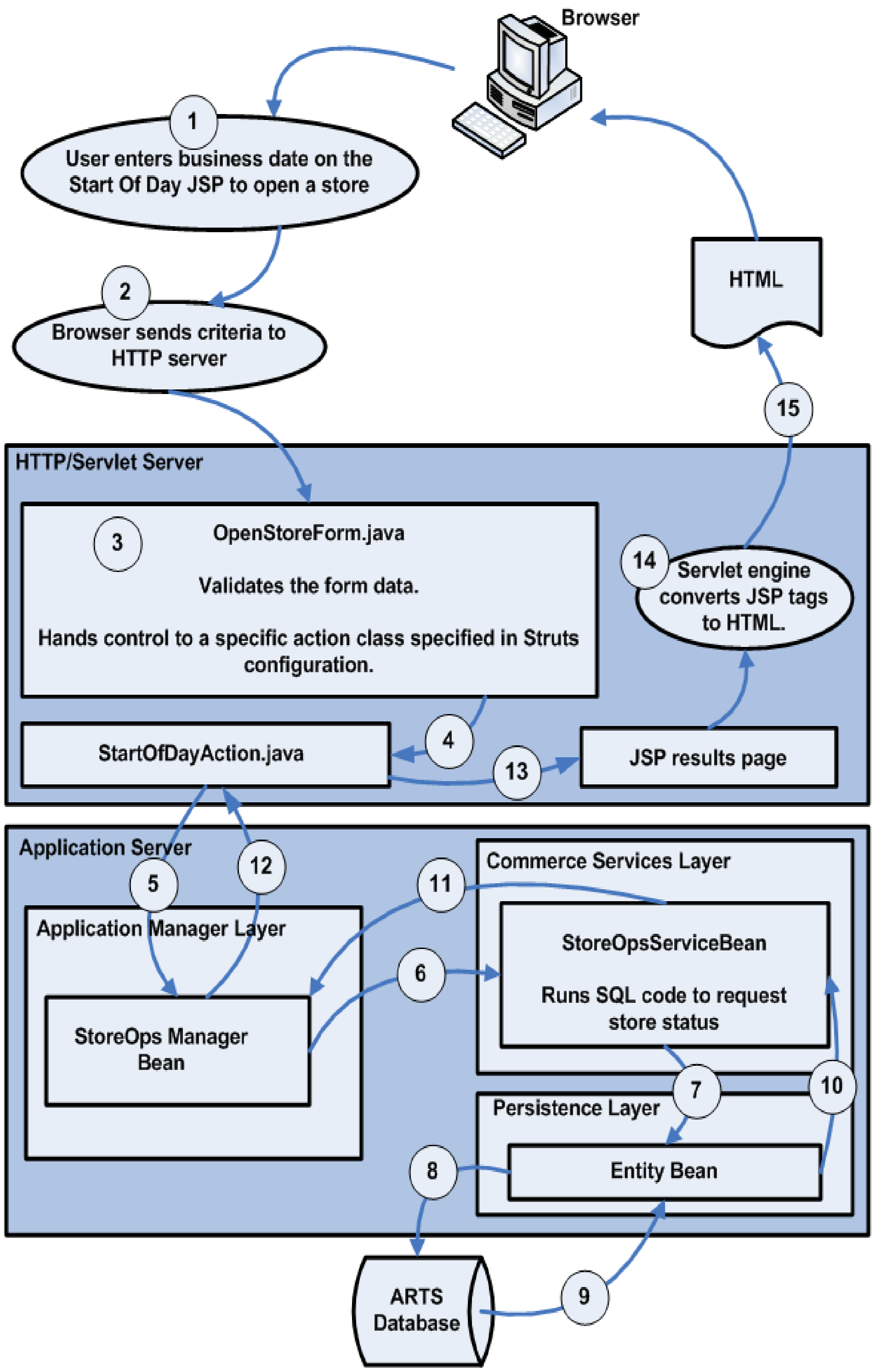
Figure 2–4 Dependencies in Back Office



Example of Operation

The following diagram describes a trip through the Back Office architecture, starting from a user's request for specific information and following through until the system's response is returned to the user's browser.

Figure 2–5 Operation of Back Office


Browser
1
User enters business date on the Start Of Day JSP to open a store
2
Browser sends criteria to HTTP server
HTML
15
HTTP/Servlet Server
3
OpenStoreForm.java
Validates the form data.
Hands control to a specific action class specified in Struts configuration.
14
Servlet engine converts JSP tags to HTML.
4
StartOfDayAction.java
13
JSP results page
Application Server
5
12
11
Commerce Services Layer
Application Manager Layer
StoreOpsServiceBean
Runs SQL code to request store status
6
StoreOps Manager Bean
7
10
Persistence Layer
Entity Bean
8
ARTS Database
9

3 Extracting Source Code

Much of this guide deals with the structure and function of Oracle Retail Back Office code, and how you can modify and extend it to serve the needs of your organization. Retailers who wish to make changes to the product, which require the source code, should contact their Oracle representative. All source code distribution requests must be approved, following Oracle's standard Source Code distribution policy.

The source code is downloadable in a single .zip file.

This .zip file contains the following:

File Name	**Comments**
cmnotes.txt	Configuration Management notes. Describe how to set up and build the source.
ORBO-<*release_number*>_source.zip	The Oracle Retail Back Office source
ORSSS-<*release_number*>_data_model.zip	Data Model (database schema) documentation
README.html	

Using pkzip, WinZip or similar utilities, you can extract ORBO-<*release_number*> onto your local hard disk. Choose the option to preserve the directory structure when you extract. Then all the source files will be placed under some directory like the following:

```
<Path to disk root>/ORBO-<release_number>_source
```

From this point on, this directory is referred to as:

```
<BO_SRC_ROOT>
```

The following is the first-level directory structure under <BO_SRC_ROOT>:

Directory	**Comments**
applications	This has one sub-directory backoffice, which contains Oracle Retail Back Office-specific code. Other directories contain code that might be common to Oracle Retail Stores applications like Central Office or Point-Of-Service.
build	Files used to compile, assemble and run functional tests.
clientinterfaces	Interface definitions, between different code modules.
commerceservices	Commerce Services code – see Chapter 9, "Commerce Services" for more details

Directory	Comments
modules	A collection of various code modules, some of which are the foundation for Commerce Services. The utility module contains SQL files used for database creation and pre-loading.
thirdparty	Executables (mostly .jar files) from third-party providers.
webapp	Web-based user interface code. Also contains the Application Managers.

In subsequent chapters, all pathnames of a code file are made relative to one of these directories. You must prepend <BO_SRC_ROOT> to the file path, to get its actual location on disk.

4 Development Environment

This chapter describes how to set up a single-user development environment for Oracle Retail Back Office. The setup enumerates the files, tools, and resources necessary to build and run the Back Office application.

When you complete the steps in this chapter, you will have a local development workspace with the ability to build the application, and an application server installation to which you can deploy the Back Office application.

This chapter assumes that you are using Oracle Application Server and the Oracle database; together, they form the officially supported platform for the current release of Oracle Retail Back Office.

Your development environment can use different tools, and you can develop variations on this procedure. Specific property file settings, in particular, might need to be modified in your environment.

For more information about product versions, see the Oracle Retail Back Office Installation Guide.

Using the Apache Ant Build Tool

Oracle Retail uses the Apache Ant build tool to compile and build executable products from source. Ant uses build information defined in various build.xml files, which in turn read from properties files. Each top-level directory in the product's source contains a build.xml file that specifies a variety of targets, or build tasks, for use by Ant.

Since each code module depends on other modules, the top-level build directory has a build.xml file which contains targets designed to build the entire system. You can build modules individually, if you build them in the correct dependency order.

Properties files (such as build.properties) contain values that are used by Ant when Ant processes tasks. Individual properties can exist in multiple files. The first setting processed by Ant is the one that is used; properties are like constants which cannot be changed once set.

If your system does not already have Ant, you can use the version shipped with Oracle Retail Back Office located at:

```
thirdparty\apache-ant-1.6.2
```

Note: Make sure that the Ant bin directory is included in your workstation's PATH.

Prerequisites for the Development Environment

The following software resources must be installed and configured before you set up the Back Office development environment as described in the following section. Where a software version is specified, use only the specified version.

- The Back Office source code, on a local (or network) hard disk. See Chapter 3, "Extracting Source Code" for details on how to extract the code.
- A database server and database. You should have access to the database server; you need its connection URL, user name and password. Depending on your organization's preferences, you might need to install the database server yourself, have a qualified database administrator to install it for you, or you can access a database server installed on another machine. The instructions in this chapter work for a local or remote database.
- JDK 1.5. Downloads and instructions are available at `http://sun.com`.

  The JAVA_HOME environment variable needs to be set in your operating system and the `%JAVA_HOME%\bin` directory needs to be added to the path.

Install Oracle Application Server

Install Oracle Application Server (OAS) under any directory you choose. Follow the instructions that come with the Application Server product. This chapter refers to this directory as <OAS_ROOT>.

Build the Back Office Application

1. CD to the Back Office build directory.
2. Edit setenv.sh (or setenv.bat on Windows). Make sure that ANT_HOME is set correctly for your system.
3. Execute setenv.sh (or setenv.bat).
4. Run the following:

   ```
   ant -Denv=backoffice clean.build.packaging
   ```

   This command will take several minutes to execute. If successful, it puts the J2EE-compatible .ear file in `applications/backoffice/assemble/assemble.working.dir/backoffice.ear`.
5. Obtain the installer in `install/dist/ORBO-<`*release_number*`>.zip`
6. Run the installer. See the *Oracle Retail Back Office Installation Guide* for more information.

5 Coding Your First Feature

This chapter describes how to modify an existing feature of Back Office using a specific example based on modifying a search page in the application's Web-based user interface. The modification of the existing search criteria page will allow it to search using additional criteria.

Note: Before extending the Back Office application, make sure the changes will not affect the PABP settings.

For more information on PABP, see the *Oracle Retail Strategic Store Solutions Security Implementation Guide*. The guide is available on Metalink:

Metalink Note: 567438.1

Related Materials

See "Example of Operation" in Chapter 2 for a diagram that shows a typical request and response flow through the Back Office architecture. The example shows the flow of data in the system while opening the store.

Before You Begin

Before you attempt to develop new Back Office code, set up your development environment as described in the preceding chapter. Verify that you can successfully build and deploy an .ear file after installing the application.

Extending Transaction Search

This section explores the modification of employee search features through the modification of the existing criteria page. The changes required to implement this functionality interact with the user interface and the internals of the Back Office system. This example takes you through the process of modifying the search criteria page and making the necessary changes to the remaining system components.

Note: Paths in this chapter are assumed to start from your local source code tree, checked out from the source code control system.

Search by Login ID

The existing employee search page allows the operator to search for employees by their employee ID, or by first and last name, or by role. This example will add the ability to search for employees by their login ID.

This example shows how:

- A user interface can be modified
- Search criteria is collected from the end user
- Data is handed off from one layer of the interface to another
- SQL queries are handled and modified

The following procedures offer general steps followed by specific examples.

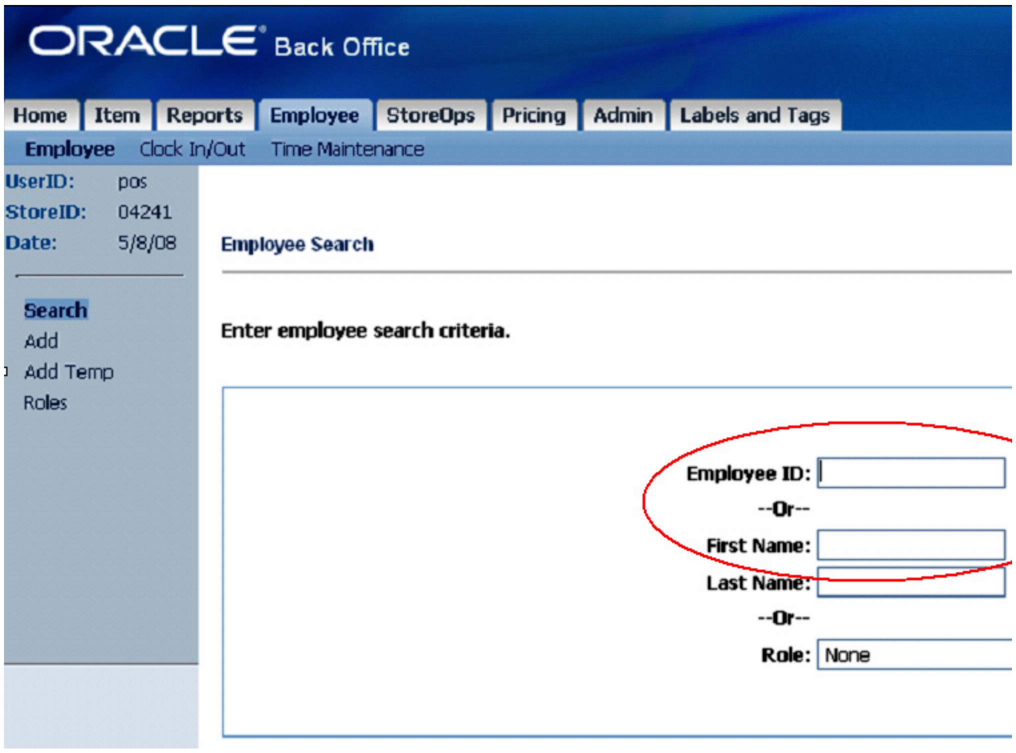
Web UI Framework

The user interface changes require that you update the JSP page to add the additional search criteria. You also need to ensure any strings you use are properly externalized for future localization. Depending on what kind of form is used you may have additional work to perform. A review of the Struts action mapping used for employee searches will reveal what changes are required to any action form in use. Finally, the action used to search will also need to be modified.

Modify the JSP File

Modification of the JSP file is fairly straightforward. Taking the existing JSP structure into consideration, insert the new search criteria between the Employee ID field and the First and Last Name fields.

Figure 5–1 Employee Search Screen



UserID: pos
StoreID: 04241
Date: 5/8/08

Employee Search

Enter employee search criteria.

Employee ID:
--Or--
First Name:
Last Name:
--Or--
Role: None

Adding the search criteria requires that you insert the appropriate HTML and JSP tags to the `/webapp/employee-webapp/web/employee/employeeSearch.jsp` file. You can identify where the changes need to be placed by looking for the message tag that displays the `employee.employeeForm.employeeFirstName.label` message. The code inthe following example presents the required changes to the employee search page.

Example 5–1 employeeSearch.jsp Modifications

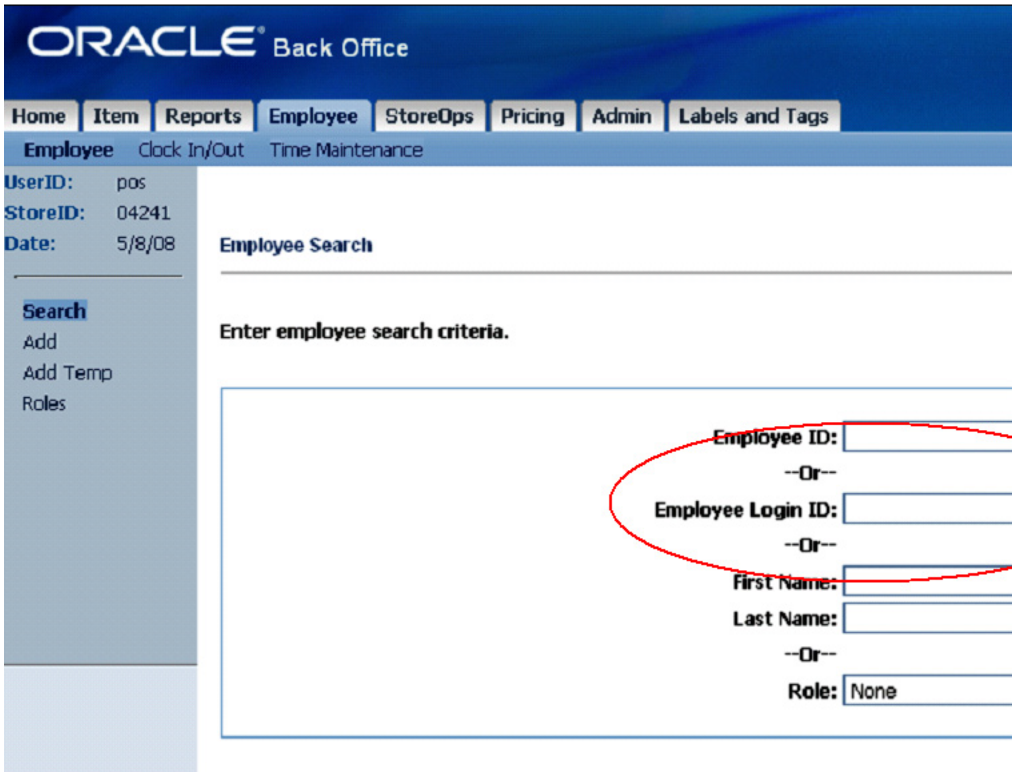
```
<tr>
    <td align="right" class="fieldname">--<bean:message
        key="prompt.or"/>--</td>
    <td align="right">&nbsp;</td>
</tr>
<tr>
    <td align="right" class="fieldname"><bean:message
        key="employee.employeeForm.employeeLoginId.label"/>: </td>
    <td align="left">
        <html:text styleClass="data"
            property="searchEmployeeLoginId"
            size="20" maxlength="10" tabindex="2"/>
    </td>
</tr>
<tr>
    <td align="right" class="fieldname">--<bean:message
        key="prompt.or"/>--</td>
    <td align="right">&nbsp;</td>
</tr>
<tr>
    <td align="right" class="fieldname"><bean:message
        key="employee.employeeForm.employeeFirstName.label"/>: </td>
    <td align="left">
        <html:text styleClass="data"
            property="searchEmployeeFirstName" size="20"
            maxlength="16" tabindex="3"/></td>
</tr>
```

The text in bold in this example is the new text that was added to the employeeSearch.jsp page. It introduces a new label and new text box to collect the new search criteria.

Note: The tab index values were incremented for all of the remaining input fields.

The modified screen is presented in the following figure.

Figure 5–2 Modified Employee Search Screen



Externalize Strings

It is very important that all user-visible strings be externalized for localization. Check to see if the label text is already defined or if you need to create it. If it exists, it is likely already in the `/webapp/i18n-webapp/src/ui/com/_360commerce/webmodules/i18n/employee.properties` file. A quick examination of this file reveals that the label text already exists and there is nothing to add.

Action Mapping

The Struts action mapping defines the important details you need to know about the code that executes the employee search when the search screen submits its data. The following example contains the XML fragment that defines the action. It comes from the `struts-employee_actions.xml` file that is included in the application's `struts-config.xml` file.

Example 5–2 Action Definition from struts-employee_actions.xml

```
<action path="/employee/searchEmployee"
    type="com._360commerce.webmodules.employee.ui.SearchEmployeeAction"
    name="employeeForm"
    scope="request"
    input="/employee/searchEmployeeView.do"
    validate="true">
    <forward name="success" path="employee.searchEmployeeViewDetails"/>
    <forward name="showEmployeeList" path="employee.searchEmployeeView"/>
    <forward name="failure" path="employee.searchEmployeeView"/>
</action>
```

You may need to modify the action form this action uses to transfer the data from the modified JSP to the aciton class. The XML fragment indicates that the form is valid for a single request and is named **employeeForm**.

Action Form

The definition of the action form is contained in another XML file, `struts-employee_forms.xml`. It is presented in the following example.

Example 5–3 Action Form Definition from struts-employee_forms.xml

```
<form-bean name="employeeForm"
type="org.apache.struts.validator.DynaValidatorForm">
    <form-property name="searchEmployeeId" type="java.lang.String"/>
    <form-property name="searchEmployeeLoginId" type="java.lang.String"/>
    <form-property name="searchEmployeeFirstName" type="java.lang.String"/>
    <form-property name="searchEmployeeLastName" type="java.lang.String"/>
    <form-property name="employeeName" type="java.lang.String"/>
    <form-property name="employeeFormattedName" type="java.lang.String"/>
    <form-property name="employeeFirstName" type="java.lang.String"/>
    <form-property name="employeeMiddleName" type="java.lang.String"/>
    <form-property name="employeeLastName" type="java.lang.String"/>
    <form-property name="employeeId" type="java.lang.String"/>
    <form-property name="employeeAlternateId" type="java.lang.String"/>
    <form-property name="employeeLoginId" type="java.lang.String"/>
    <form-property name="employeeRole" type="java.lang.String"/>
    <form-property name="employeeStatus" type="java.lang.String"/>
    <form-property name="socialSecurityNumber" type="java.lang.String"/>
    <form-property name="employeeStatusCode" type="java.lang.String"/>
    <form-property name="workGroupId" type="java.lang.String"/>
    <form-property name="employeeLocale" type="java.lang.String"/>
    <form-property name="groupID" type="java.lang.String"/>
    <form-property name="employeeValidity" type="java.lang.String"/>
    <form-property name="employeeType" type="java.lang.String"/>
    <form-property name="employeeActualStatusCode" type="java.lang.String"/>
    <form-property name="errorMessage" type="java.lang.String"/>
    <form-property name="employeeStoreId" type="java.lang.String"/>
    <form-property name="employeeRoleName" type="java.lang.String"/>
</form-bean>
```

The definition reveals that the employee search screen is using a DynaValidatorForm provided by Struts. All of the form properties are defined here in the XML. The line in bold was added to introduce our new search criteria. Validation of the entered data should be addressed by proper configuration of the Struts validator. Examples of how to validate form data using the Struts DynaValidatorForm can be found in the `struts-employee_validator.xml` file provided with the source code and the Struts documentation.

Action Modification

With the new search critera added to the action form, we can now turn our attention to modifying the action class that actually performs the search from the user interface. The action class used is located at `/webapp/employee-webapp/src/ui/com/_360commerce/webmodules/employee/ui/SearchEmployeeAction.java`. The action class `execute()` method is broken into three sections using an **if-then-else** statement. The following example provides a portion of the updates required to enable the action to search for employees based on their login ID.

Example 5–4 Modifications to SearchEmployeeAction.java

```
public class SearchEmployeeAction extends Action
{
        public ActionForward execute(...) throws Exception
    {
...
        String employeeLoginId = "";
...
        try
        {
...
            employeeLoginId = dynaActionForm
                .get("searchEmployeeLoginId").toString();
...
            if (!employeeId.equals(""))...
            else if (!employeeLoginId.equals(""))
            {
                EmployeeDTO employeeDTO = employeeManager
                    .getEmployeeByLoginID(employeeLoginId);
                -- similar code to the first if condition to prepare
                -- the necessary data for display to the user since
                -- this type of search will match only a one employee
            }
            else if (employeeId.equals("")
                && !(employeeFirstName.equals("")
                && employeeLastName.equals("")))...
            else...
        }
...
    }
}
```

Because the only differences between the search by employee ID and search by employee login ID are how the employee is found, the code in the first two **if** blocks are almost identical and could be refactored to share implementations. For the purposes of this example that repated code and possible refactoring has been omitted. The important difference is the call to the `employeeManager.getEmployeeByLoginID()` method.

Application Services

The Employee Manager session bean already contained the required method to search for employees by their login ID. The business interface for the employee manager is located in the file `/webapp/employee-webapp/src/app/com/_360commerce/webmodules/employee/app/EmployeeManagerIfc.java`. It contains the declarations of the methods available to the user interface. The method to find an employee by their login ID is presented in the following example.

Example 5–5 EmployeeManagerIfc.java

```
    /**
     * Finds the employee with the specified Login ID.
     *
     * @param loginID the ID of the employee to find.
     * @return A DTO containing the employee data.
     * @throws EmployeeNotFoundException if the employee cannot be
     * found
     */
    EmployeeDTO getEmployeeByLoginID(String loginID)
```

```
        throws EmployeeNotFoundException, RemoteException;
```

The bean implementation is located in `/webapp/employee-webapp/src/app/com/_360commerce/webmodules/employee/app/ejb/EmployeeManagerBean.java` As a façade it simply delegates the call to the Employee Service bean in the Commerce Services layer.

Commerce Services

The Employee Service bean in the Commerce Services layer is another façade that prevents direct access to the entity beans. The implementation obtains an instance of the EmployeeLocalHome and invokes the findByLoginID() method. Examination of the Employee entity bean will reveal that it is directly querying the database to find the user with the corresponding login ID.

Other Examples

The *Oracle Retail Central Office Operations Guide* contains a similar, but more detailed, example of how to add an entirely new page and supporting objects. It is recommended that readers also review that example as well for additional information.

6 General Development Standards

The standards in this chapter have been adopted by Oracle Retail product and service development teams. These standards are intended to reduce bugs and increase the quality of the code. The chapter covers basic standards, architectural issues, and common frameworks. These guidelines apply to all Oracle Retail applications.

Basics

The guidelines in this section cover common coding issues and standards.

Java Recommendations

The following dos and don'ts are guidelines for what to avoid when writing Java code.

- Do use polymorphism.
- Do have only one return statement per function or method; make it the last statement.
- Do use constants instead of literal values when possible.
- Do import only the classes necessary instead of using wildcards.
- Do define constants at the top of the class instead of inside a method.
- Do keep methods small, so that they can be viewed on a single screen without scrolling.
- Do not have an empty catch block. This destroys an exception from further down the line that might include information necessary for debugging.
- Do not concatenate strings. Oracle Retail products tend to be string-intensive and string concatenation is an expensive operation. Use StringBuffer or StringBuilder instead.
- Do not use function calls inside looping conditionals (for example, `while (i <=name.len())`). This calls the function with each iteration of the loop and can affect performance.
- Do not use a static array of strings.
- Do not use public attributes.
- Do not use a switch to make a call based on the object type.

Avoiding Common Java Bugs

Fatal Java bugs are not found at compile time and are not easily found at runtime.

Table 6–1 lists bugs that can be avoided and their preventative-measure recommendations.

Table 6–1 **Common Java Bugs**

Bug	Preventative Measure
null pointer exception	Check for null before using an object returned by another method.
boundary checking	Check the validity of values returned by other methods before using them.
array index out of bounds	When using a value as a subscript to access an array element directly, first verify that the value is within the bounds of the array.
incorrect cast	When casting an object, use instanceof to ensure that the object is of that type before attempting the cast.

Formatting

Follow these formatting standards to ensure consistency with existing code.

Note: A code block is defined as a number of lines proceeded with an opening brace and ending with a closing brace.

- Indenting/braces—Indent all code blocks with four spaces (not tabs). Put the opening brace on its own line following the control statement and in the same column. Statements within the block are indented. Closing brace is on its own line and in same column as the opening brace. Follow control statements (`if`, `while`, and so forth) with a code block with braces, even when the code block is only one line long.
- Line wrapping—If line breaks are in a parameter list, line up the beginning of the second line with the first parameter on the first line. Lines should not exceed 120 characters.
- Spacing—Include a space on both sides of binary operators. Do not use a space with unary operators. Do not use spaces around parenthesis. Include a blank line before a code block.
- Deprecation—Whenever you deprecate a method or class from an existing release, mark it as deprecated, noting the release in which it was deprecated, and what methods or classes should be used in place of the deprecated items; these records facilitate later code cleanup.
- Header—The file header should include the tag for revision and log history.

Example 6–1 Header Sample

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

    Copyright (c) 1998-2008 Oracle Retail, Inc.    All Rights Reserved.

    $Log$

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
package com._360commerce.samples;

// Import only what is used
import Java.math.BigDecimal;
import com._360commerce.common.utility.Util;
```

```
/**
     This class is a sample class. Its purpose is to illustrate proper
     formatting.
     @version $Revision$
**/
public class Sample extends AbstractSample
implements SampleIfc
{
    // revision number supplied by configuration management tool
    public static String revisionNumber = "$Revision$";
    // This is a sample data member.
    // Use protected access since someone may need to extend your code.
    // Initializing the data is encouraged.
    protected String sampleData = "";

    /**
        Constructs Sample object.
        Include the name of the parameter and its type in the javadoc.
        @param initialData String used to initialize the Sample.
    **/
    public Sample(String initialData)
    {
        sampleData = initialData;
        // Declare variables outside the loop
        int length = sampleData.length();
        BigDecimal[] numberList = new BigDecimal[length];

        // Precede code blocks with blank line and pertinent comment
        for (int i = 0; i < length; i++)
        {
            // Sample wrapping line.
            numberList[i] = someInheritedMethodWithALongName(Util.I_BIG_DECIMAL_
ONE,
 sampleData,
 length - i);
        }
    }
}
```

Javadoc

- Make code comments conform to Javadoc standards.
- Include a comment for every code block.
- Document parameters and return codes for every method, and include a brief statement as to the method's purpose.

Naming Conventions

Names should not use abbreviations except when they are widely accepted within the domain (such as the customer abbreviation, which is used extensively to distinguish customized code from product code).

Table 6–2 lists additional naming conventions.

Table 6–2 **Naming Conventions**

Element	Description	Example
Package Names	Package names are entirely lower case and should conform to the documented packaging standards.	com.extendyourstore.packagename com.mbs.packagname
Class Names	Mixed case, starting with a capital letter. Exception classes end in Exception; interface classes end in Ifc; unit tests append Test to the name of the tested class.	DatabaseException DatabaseExceptionTest FoundationScreenIfc
File Names	File names are the same as the name of the class.	DatabaseException.java

Method Names	Method names are mixed case, starting with a lowercase letter. Method names are an action verb, where possible. Boolean-valued methods should read like a question, with the verb first. Accessor functions use the prefixes get or set.	isEmpty()
		hasChildren()
		getAttempt()
		setName()

Attribute Names	Attribute names are mixed case, starting with a lowercase letter.	lineItemCount
Constants	Constants (static final variables) are named using all uppercase letters and underscores.	final static int NORMAL_SIZE = 400
EJBs — entity	Use these conventions for entity beans, where *Transaction* is a name that describes the entity.	TransactionBean TransactionIfc TransactionLocal TransactionLocalHome TransactionRemote TransactionHome
EJBs — session	Use these conventions for session beans, where *Transaction* is a name that describes the session.	TransactionService TransactionAdapter TransactionManager

SQL Guidelines

The following general guidelines apply when creating SQL code:

- Keep SQL code out of client/UI modules. Such components should not interact with the database directly.
- Table and column names must be no more than 18 characters.
- Comply with ARTS specifications for new tables and columns. If you are creating something not currently specified by ARTS, strive to follow the ARTS naming conventions and guidelines.
- Document and describe every object, providing both descriptions and default values so that we can maintain an up-to-date data model.
- Consult your data architect when designing new tables and columns.

- Whenever possible, avoid vendor-specific extensions and strive for SQL-92 compliance with your SQL.
- All SQL commands should be uppercase because the DataFilters currently only handle uppercase.
- If database-specific code is used in the source, move it into the JdbcHelpers.
- All JDBC operations classes must be thread-safe.

Do the following to avoid errors:

- Pay close attention when cutting and pasting SQL.
- Always place a carriage return at the end of the file.
- Test your SQL before committing.

The subsections that follow describe guidelines for specific database environments.

DB2 SQL

Table 6–3 shows examples of potential problems in DB2 SQL code.

Table 6–3 **DB2 SQL Code Problems**

Problem	Problem Code	Corrected Code

Don’t use quoted integers or unquoted char and varchar values; these cause DB2 to produce errors.	`CREATE TABLE BLAH`
	`(`
	` FIELD1 INTEGER,`
	` FIELD2 CHAR(4)`
	`);`
	`INSERT INTO BLAH (FIELD1, FIELD2) VALUES (`**`'5'`**`, `**`1020`**`);`
		`CREATE TABLE BLAH`
		`(`
		` FIELD1 INTEGER,`
		` FIELD2 CHAR(4)`
		`);`
		`INSERT INTO BLAH (FIELD1, FIELD2)`
		`VALUES (5, '1020');`

Don’t try to declare a field default as NULL.	`CREATE TABLE BLAH`
	`(`
	` FIELD1 INTEGER `**`NULL`**`,`
	` FIELD2 CHAR(4) NOT NULL`
	`);`
		`CREATE TABLE BLAH`
		`(`
		` FIELD1 INTEGER,`
		` FIELD2 CHAR(4) NOT NULL`
		`);`

Oracle SQL

Table 6–4 provides some examples of common syntax problems which cause Oracle to produce errors.

Table 6–4 **Oracle SQL Code Problems**

Problem	Problem Code	Corrected Code

Blank line in code block causes error.	`CREATE TABLE BLAH`
	`(`
	` FIELD1 INTEGER,`
	` FIELD2 VARCHAR(20)`
	
	`);`
		`CREATE TABLE BLAH`
		`(`
		` FIELD1 INTEGER,`
		` FIELD2 VARCHAR(20)`
		`);`

When using NOT NULL with a default value, NOT NULL must follow the DEFAULT statement.	`CREATE TABLE BLAH`
	`(`
	` FIELD1 INTEGER` **`NOT NULL`** `DEFAULT`
	`0,`
	` FIELD2 VARCHAR(20)`
	`);`
		`CREATE TABLE BLAH`
		`(`
		` FIELD1 INTEGER DEFAULT 0` **`NOT NULL`**`,`
		` FIELD2 VARCHAR(20)`
		`);`

In a CREATE or INSERT, do not place a comma after the last item.	`CREATE TABLE BLAH`
	`(`
	` FIELD1 INTEGER,`
	` FIELD2 VARCHAR(20),`
	`);`
		`CREATE TABLE BLAH`
		`(`
		` FIELD1 INTEGER,`
		` FIELD2 VARCHAR(20)`
		`);`

Unit Testing

Some general notes apply:

- Break large methods into smaller, testable units.
- Although unit testing might be difficult for tour scripts, apply it for Java components within the Point-of-Service code.
- If you add a new item to the codebase, make sure your unit tests prove that the new item can be extended.
- In unit tests, directly create the data or preconditions necessary for the test (in a `setup()` method) and remove them afterwards (in a `teardown()` method). JUnit expects to use these standard methods in running tests.

Architecture and Design Guidelines

This section provides guidelines for making design decisions which are intended to promote a robust architecture.

AntiPatterns

An AntiPattern is a common solution to a problem which results in negative consequences. The name contrasts with the concept of a pattern, a successful solution to a common problem.

Table 6–5 lists AntiPatterns that can introduce bugs and reduce the quality of code.

Table 6–5 **Common AntiPatterns**

Pattern	Description	Solution
Reinvent the Wheel	Sometimes code is developed in an unnecessarily unique way that leads to errors, prolonged debugging time and more difficult maintenance.	The analysis process for new features provides awareness of existing solutions for similar functionality so that you can determine the best solution. There must be a compelling reason to choose a new design when a proven design exists. During development, a similar pattern should be followed in which existing, proven solutions are implemented before new solutions.
Copy-and-paste Programming, classes	When code needs to be reused, it is sometimes copied and pasted instead of using a better method. For example, when a whole class is copied to a new class when the new class could have extended the original class. Another example is when a method is being overridden and the code from the super class is copied and pasted instead of calling the method in the super class.	Use object-oriented techniques when available instead of copying code.
Copy-and-paste Programming, XML	A new element (such as a Site class or an Overlay XML tag) can be started by copying and pasting a similar existing element. Bugs are created when one or more pieces are not updated for the new element. For example, a new screen might have the screen name or prompt text for the old screen.	If you copy an existing element to create a new element, manually verify each piece of the element to ensure that it is correct for the new element.
Project Mismanagement/ Common Understanding	A lack of common understanding between managers, Business Analysts, Quality Assurance and developers can lead to missed functionality, incorrect functionality and a larger-than-necessary number of defects.	Before you consider code for the requirement finished, all issues must be resolved and the code must match the requirements.
Stovepipe	Multiple systems within an enterprise are designed independently. The lack of commonality prevents reuse and inhibits interoperability between systems. For example, a change to till reconcile in Back Office might not consider the impact on Point-of-Service. Another example is making a change to a field in the Oracle Retail database for a Back Office feature without handling the Point-of-Service effects.	Coordinate technologies across applications at several levels. Define basic standards in infrastructures for the suite of products. Only mission-specific functions should be created independently of the other applications within the suite.

Designing for Extension

This section defines how to code product features so that they can be easily extended. It is important that developers on customer projects whose code might be rolled back into the base product follow these standards as well as the guidelines in Chapter 7, "Extension Guidelines".

Note: Before extending the Back Office application, make sure the changes will not affect the PABP settings.

For more information on PABP, see the *Oracle Retail Strategic Store Solutions Security Implementation Guide*. The guide is available on Metalink:

Metalink Note: 567438.1

- Separate external constants such as database table and column names, JMS queue names, port numbers from the rest of the code. Store them in (in order of preference):
  - Configuration files
  - Deployment descriptors
  - Constant classes and interfaces
- Make sure the SQL code included in a component does not touch tables not directly owned by that component.
- Make sure there is some separation from DTO and ViewBean type classes so we have abstraction between the service and the presentation.
- Consider designing so that any fine-grained operation within the larger context of a coarse grain operation can be factored out in a separate algorithm class, so that the fine-grained operation can be replaced without reworking the entire activity flow of the larger operation.

Common Frameworks

This section provides guidelines which are common to the Oracle Retail applications.

Internationalization

The only language currently supported is United States English.

Oracle Retail does not provide support for any customer extensions made to the base Back Office product.

Logging

Oracle Retail's systems use Log4J for logging. When writing log commands, use the following guidelines:

- Use calls to Log4J rather than System.out from the beginning of your development. Unlike System.out, Log4J calls are naturally written to a file, and can be suppressed when desired.
- Log exceptions where you catch them, unless you are going to rethrow them. This preserves the context of the exceptions and helps reduce duplicate exception reporting.

- Logging uses few CPU cycles, so use debugging statements freely.
- Use the correct logging level:
  - FATAL—crashing exceptions
  - ERROR—nonfatal, unhandled exceptions (there should be few of these)
  - INFO—life cycle/heartbeat information
  - DEBUG—information for debugging purposes

The following sections provide additional information on guarding code, when to log, and how to write log messages.

Guarding Code

Testing shows that logging takes up very little of a system's CPU resources. However, if a single call to your formatter is abnormally expensive (stack traces, database access, network IO, large data manipulations, and so forth), you can use Boolean methods provided in the Logger class for each level to determine whether you have that level (or better) currently enabled; Jakarta calls this a code guard:

Example 6–2 Wrapping Code in a Code Guard

```
    if (log.isDebugEnabled()) {
        log.debug(MassiveSlowStringGenerator().message());
    }
```

An interesting use of code guards, however, is to enable debug-only code, instead of using a DEBUG flag. Using Log4J to maintain this functionality lets you adjust it at runtime by manipulating Log4J configurations.

For instance, you can use code guards to simply switch graphics contexts in your custom swing component:

Example 6–3 Switching Graphics Contexts via a Logging Level Test

```
protected void paintComponent(Graphics g) {

    if (log.isDebugEnabled()) {
        g = new DebugGraphics(g, this);
    }

    g.drawString("foo", 0, 0);
}
```

When to Log

There are three main cases for logging:

- Exceptions—Should be logged at an error or fatal level.
- Heartbeat/Life cycle—For monitoring the application; helps to make unseen events clear. Use the info level for these events.
- Debug—Code is usually littered with these when you are first trying to get a class to run. If you use System.out, you have to go back later and remove them to keep. With Log4J, you can simply raise the log level. Furthermore, if problems pop up in the field, you can lower the logging level and access them.

Writing Log Messages

When Log4J is being used, any log message might be seen by a user, so the messages should be written with users in mind. Cute, cryptic, or rude messages are inappropriate. The following sections provide additional guidelines for specific types of log messages.

Exception Messages

A log message should have enough information to give the user an understanding of the problem and enable the user to fix the problem. Poor logging messages say something opaque like “load failed.”

Consider this piece of code:

```
try {
    File file = new File(fileName);
    Document doc = builder.parse(file);

    NodeList nl = doc.getElementsByTagName("molecule");
    for (int i = 0; i < nl.getLength(); i++) {
        Node node = nl.item(i);
        // something here
    }

} catch {
   // see below
}
```

and these two ways of logging exceptions:

```
} catch (Exception e){
    log.debug("Could not load XML");
}

} catch (IOException e){
    log.error("Problem reading file " + fileName, e);
} catch (DOMException e){
    log.error("Error parsing XML in file " + fileName, e);
} catch (SAXException e){
    log.error("Error parsing XML in file " + fileName, e);
}
```

In the first case, you'll get an error that just tells you something went wrong. In the second case, you're given slightly more context around the error, in that you know if you can't find it, load it, or parse it, and you're given that key piece of data: the file name.

The log lets you augment the message in the exception itself. Ideally, with the messages, the stack trace, and type of exception, you'll have enough information to be able to reproduce the problem at debug time. Given that, the message can be reasonably verbose.

For instance, the `fail()` method in JUnit really just throws an exception, and whatever message you pass to it is in effect logging. It’s useful to construct messages that contain a great deal of information about what you are looking for:

Example 6–4 JUnit

```
if (! list.contains(testObj)) {

    StringBuffer buf = new StringBuffer();
    buf.append("Could not find object " + testObj + " in list.\n");
    buf.append("List contains: ");
    for (int i = 0; i < list.size(); i++) {
        if (i > 0) {
             buf.append(",");
        }
        buf.append(list.get(i));
    }
    fail(buf.toString());
}
```

Heartbeat or Life cycle Messages

The log message should succinctly display what portion of the life cycle is occurring (login, request, loading, and so forth) and what apparatus is doing it (is it a particular EJB, are there multiple servers running, and so forth)

These message should be fairly terse, since you expect them to be running all the time.

Debug Messages

Debug statements are going to be your first insight into a problem with the running code, so having enough, of the right kind, is important.

These statements are usually either of an intra-method-life cycle variety:

```
log.debug("Loading file");

File file = new File(fileName);
log.debug("loaded.  Parsing...");
Document doc = builder.parse(file);
log.debug("Creating objects");
for (int i ...
```

or of the variable-inspection variety:

```
log.debug("File name is " + fileName);

log.debug("root is null: " + (root == null));
log.debug("object is at index " + list.indexOf(obj));
```

Exception Handling

The following are the key guidelines for exception handling:

- Handle the exceptions that you can (File Not Found, and so forth).
- Fail fast if you can't handle an exception.
- Log every exception with Log4J, even when first writing the class, unless you are rethrowing the exception.
- Include enough information in the log message to give the user or developer a chance to know what went wrong.
- Nest the original exception if you rethrow one.

Types of Exceptions

The EJB specification divides exceptions into the following categories:

JVM Exceptions

You cannot recover from these; when one is thrown, it's because the JVM has entered a kernel panic state that the application cannot be expected to recover from. A common example is an Out of Memory error.

System Exceptions

Similar to JVM exceptions, these are generally, though not always, non-recoverable exceptions. In the commons-logging parlance, these are *unexpected* exceptions. The canonical example here is NullPointerException. The idea is that if a value is null, often you don't know what you should do. If you can simply report back to your calling method that you got a null value, do that. If you cannot gracefully recover, say from an IndexOutOfBoundsException, treat as a system exception and fail fast.

Application Exceptions

These are the expected exceptions, usually defined by specific application domains. It is useful to think of these in terms of recoverability. A FileNotFoundException is sometimes easy to rectify by simply asking the user for another file name. But something that's application specific, like JDOMException, still might not be recoverable. The application can recognize that the XML it is receiving is malformed, but it still might not be able to do anything about it.

Avoid java.lang.Exception

Avoid throwing the generic Exception; choose a more specific (but standard) exception.

Avoid Custom Exceptions

Custom exceptions are rarely needed. The specific type of exception thrown is rarely important; don't create a custom exception if there is a problem with the formatting of a string (ApplicationFormatttingException) instead of reusing IllegalArgumentException.

The best case for writing a custom exception is if you can provide additional information to the caller which is useful for recovering from the exception or fixing the problem. For example, the JPOSExceptions can report problems with the physical device. An XML exception could have line number information embedded in it, allowing the user to easily detect where the problem is. Or, you could subclass NullPointer with a little debugging magic to tell the user what method or variable is null.

Catching Exceptions

The following sections provide guidelines on catching exceptions.

Keep the Try Block Short The following example, from a networking testing application, shows a loop that was expected to require approximately 30 seconds to execute (since it calls `sleep(3000)` ten times):

Example 6–5 Network Test

```
    for (int i = 0; i < 10; i++) {
        try {
            System.out.println("Thread " + Thread.currentThread().getName() + "
requesting number " + i);
```

```
            URLConnection con = myUrl.openConnection();
            con.getContent();
            Thread.sleep(3000);
        } catch (Exception e) {
            log.error("Error getting connection or content", e);
        }
    }
```

The initial expectation was for this loop to take approximately 30 seconds, since the `sleep(3000)` would be called ten times. Suppose, however, that `con.getContent()` throws an IOException. The loop then skips the `sleep()` call entirely, finishing in 6 seconds. A better way to write this is to move the `sleep()` call outside of the try block, ensuring that it is executed:

Example 6–6 Network Test with Shortened Try Block

```
    for (int i = 0; i < 10; i++) {

        try {
            System.out.println("Thread " + Thread.currentThread().getName() + "
requesting number " + i);
            URLConnection con = myUrl.openConnection();
            con.getContent();
        } catch (Exception e) {
            log.error("Error getting connection or content", e);
        }
        Thread.sleep(3000);
    }
```

Avoid Throwing New Exceptions When you catch an exception, then throw a new exception in its place, you replace the context of where it was thrown with the context of where it was caught.

A slightly better way is to throw a wrapped exception:

Example 6–7 Wrapped Exception

```
try {
    Class k1 = Class.forName(firstClass);
    Class k2 = Class.forName(secondClass);
    Object o1 = k1.newInstance();
    Object o2 = k2.newInstance();

} catch (Exception e) {
   throw new MyApplicationException(e);
}
```

However, the onus is still on the user to call `getCause()` to see what the real cause was. This makes most sense in an RMI-type environment, where you need to tunnel an exception back to the calling methods.

A better way than throwing a wrapped exception is to simply declare that your method throws the exception, and let the caller figure it out:

Example 6–8 Declaring an Exception

```
public void buildClasses(String firstName, String secondName)
       throws InstantiationException, ... {

       Class k1 = Class.forName(firstClass);
       Class k2 = Class.forName(secondClass);
       Object o1 = k1.newInstance();
       Object o2 = k2.newInstance();
}
```

However, there might be times when you want to deal with some cleanup code and then rethrow an exception:

Example 6–9 Clean Up First, then Rethrow Exception

```
try {
        someOperation();
    } catch (Exception e) {
        someCleanUp();
        throw e;
    }
```

Catching Specific Exceptions There are various exceptions for a reason: so you can precisely identify what happened by the type of exception thrown. If you just catch Exception (rather than, say, ClassCastException), you hide information from the user. On the other hand, methods should not generally try to catch every type of exception. The rule of thumb is related to the fail-fast/recover rule: catch as many different exceptions as you are going to handle.

Favor a Switch over Code Duplication The syntax of try-and-catch makes code reuse difficult, especially if you try to catch at a granular level. If you want to execute some code specific to a certain exception, and some code in common, you're left with either duplicating the code in two catch blocks, or using a switch-like procedure. The switch-like procedure, shown in the following example, is preferred because it avoids code duplication:

Example 6–10 Using a Switch to Execute Code Specific to an Exception

```
try{
    // some code here that throws Exceptions...
} catch (Exception e) {
    if (e instanceof LegalException) {
        callPolice((LegalException) e);
    } else if (e instanceof ReactorException) {
        shutdownReactor();
    }
    logException(e);
    mailException(e);
    haltPlant(e);
}
```

The following example is preferred, in these relatively rare cases, to using multiple catch blocks:

Example 6–11 Using Multiple Catch Blocks Causes Duplicate Code

```
try{
        // some code here that throws Exceptions...
    } catch (LegalException e) {
        callPolice(e);
        logException(e);
        mailException(e);
        haltPlant(e);
    } catch (ReactorException e) {
        shutdownReactor();
        logException(e);
        mailException(e);
        haltPlant(e);
    }
```

Exceptions tend to be the backwater of the code; requiring a maintenance developer, even yourself, to remember to update the duplicate sections of separate catch blocks is a recipe for future errors.

7 Extension Guidelines

This document describes the various extension mechanisms available in the Commerce Services framework. There are multiple forces driving each extension that determine the correct strategy in each case.

The product has four distinct layers of logic:

UI layer
Struts/Tiles implementation utilizing Actions for processing UI requests and JSP pages with Tiles providing layout.

Application Manager
Session facade for the UI (or external system) that models application business methods. May or may not be reusable between applications. Provides for remote accessibility.

Commerce Service
Session facade for the service that models coarse-grained business logic that should be reusable between applications.

Persistence
Entity beans that are fine-grained and consumed by the service. The entities are local to the service that controls them.

Note: Before extending the Back Office application, make sure the changes will not affect the PABP settings.

For more information on PABP, see the *Oracle Retail Strategic Store Solutions Security Implementation Guide*. The guide is available on Metalink:

Metalink Note: 567438.1

Audience

This chapter provides guidelines for extending the Oracle Retail Enterprise applications. The guidelines are designed for three audiences:

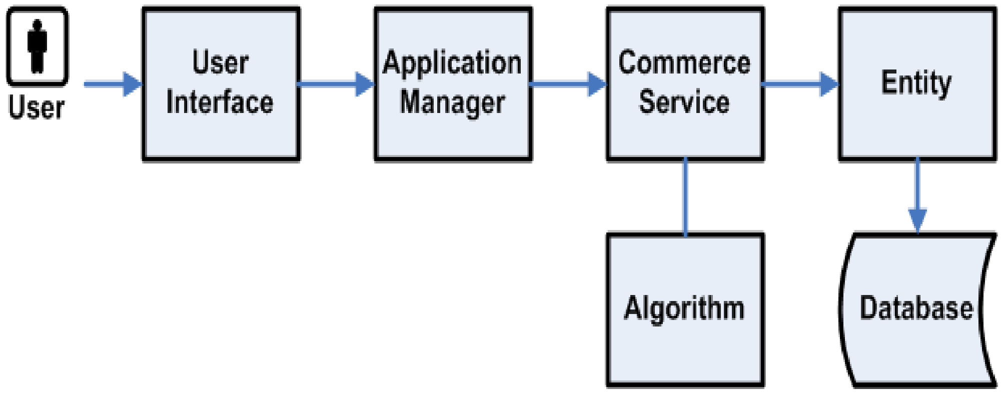
- Members of customer architecture and design groups can use this chapter as the basis for their analysis of the overall extension of the systems.
- Members of Oracle Retail's Technology and Architecture Group can use this chapter as the basis for analyzing the viability of the overall extension strategy for enterprise applications.

- Developers on the project teams can use this chapter as a reference for code-level design and extension of the product for the solution that is released.

Application Layers

The following diagram describes the general composition of the enterprise applications. The following sections describe the purpose and responsibility of each layer.

Figure 7–1 Application Layers



User Interface

The user interface (UI) framework consists of Struts Actions, Forms, and Tiles, along with Java server pages (JSPs).

- Struts configuration
- Tiles definition
- Cascading style sheets (CSS)
- JSP pages
- Resource bundles for i18N

Application Manager

The Application Manager components are coarse-grained business objects that define the behavior of related Commerce Services based on the application context.

- Session Beans
- View Beans for the UI

Commerce Service

A commerce service is a fine grained component of reusable business logic.

- Session Beans
- Data Transfer Objects (DTOs)

Algorithm

An SPI-like interface defined to enable more fine-grained pieces of business functionality to be replaced without impacting overall application logic. For reference, review the various calculator classes that are contained in "Financial Totals Service" on page 9-7.

Entity

Fine-grained entity beans owned by the commerce service. The current strategy for creating entity beans in the commerce service layer is BMP.

Database

The Oracle Retail enterprise applications support the ARTS standard database schema. The same tables referenced by Central Office and Back Office are a superset of the tables that support Point-of-Service.

Extension and Customization Scenarios

Style and Appearance Changes

This should only present minor changes to the UI layer of the application. These types of changes, while extremely common, should represent minimal impact to the operation of the product. Typical changes could be altering the style of the application (fonts/colors/formatting) or the types of messages that are displayed.

Application impact:

- Struts configuration (flow)
- Tile definition
- Style Sheet
- Minor JSP changes, such as moving fields
- Changing static text through resource bundles

Additional Information Presented to User

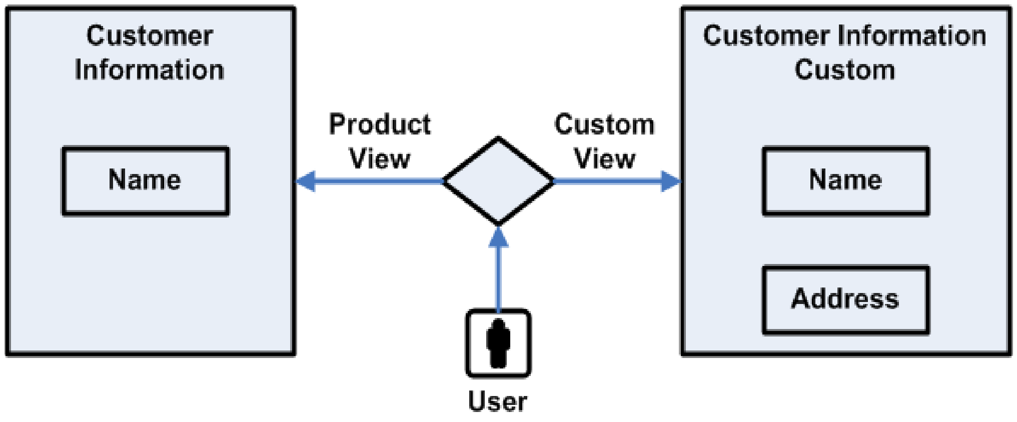
This is one of the more common extensions to the base product: enabling the full life cycle management of information required by a particular customer that is not represented in the base product.

If the information is simply presented and persisted then we can choose a strategy that simply updates the UI and persistence layers and passes the additional information through the service layer.

However, if the application must use the additional information to alter the business logic of a service, then each layer of the application must be modified accordingly.

This scenario generally causes the most pervasive changes to the system; it should be handled in a manner that can preserve an upgrade path.

Figure 7–2 Managing Additional Information



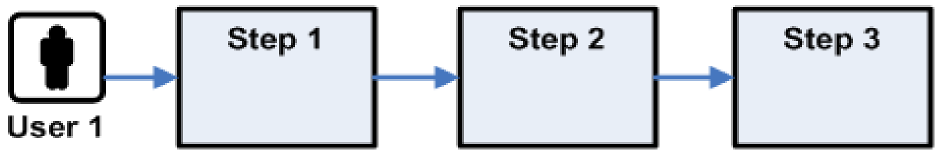
Application impact for this change:

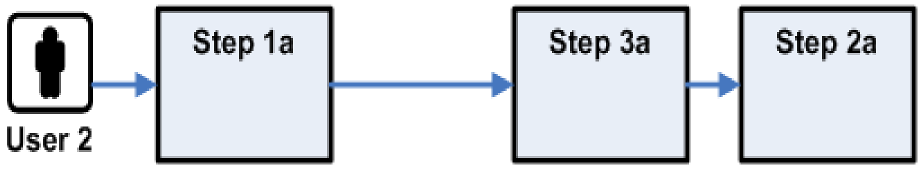
- JSP pages
- View Beans
- Struts configuration
- UI Actions
- UI Forms
- Application Manager
- Commerce Service
- Entity
- Database Schema

Changes to Application Flow

Sometimes a multi-step application flow can be rearranged or customized without altering the layers of the application outside of the UI. These changes can be accomplished by changing the flow of screens with the struts configuration.

Figure 7–3 Changing Application Flow





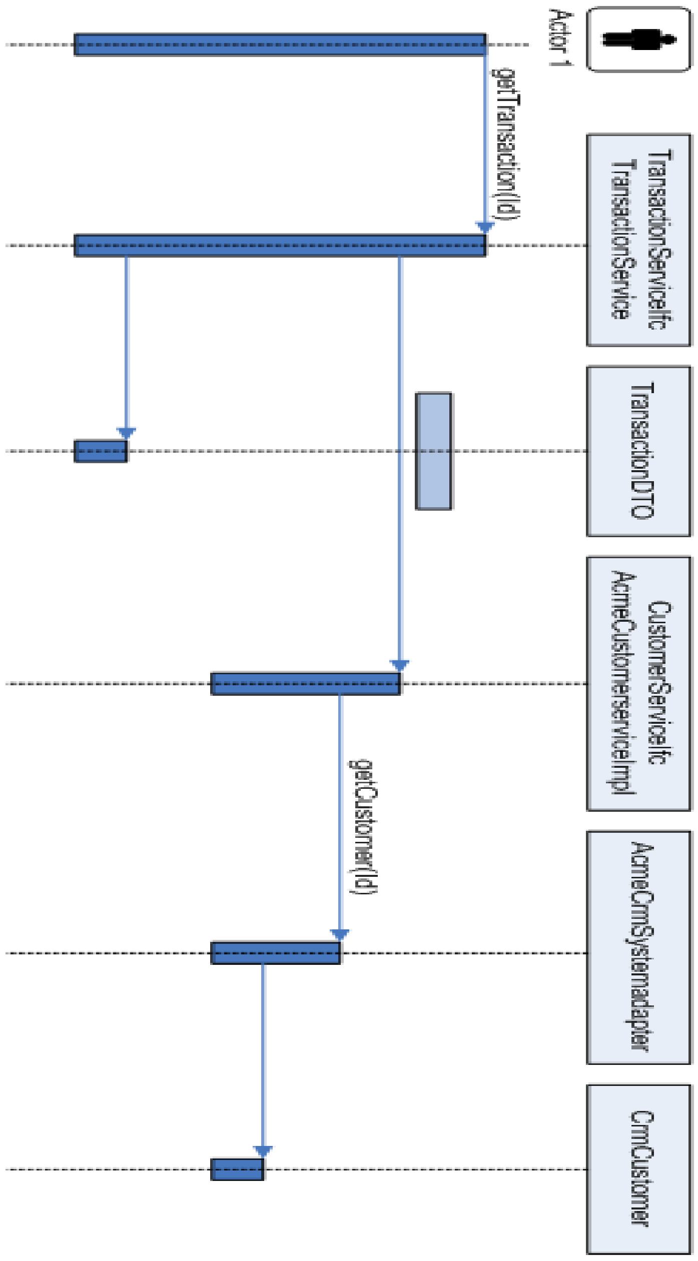
Application impact for this change:

- Struts configuration

Access Data From a Different Database

This customization describes accessing the same business data from a different database schema. No new fields are added or joined unless for deriving existing interface values. This scenario would most likely not be found isolated from the other scenarios.

Figure 7–4 Accessing Data from a Different Database



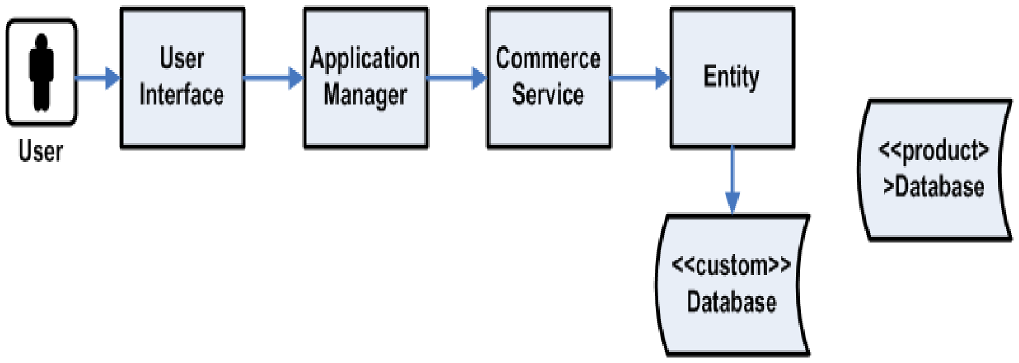
Application impact for this change:

- Entity Beans
- Database Schema

Access Data From External System

This customization involves replacing an entire Commerce Service with a completely new implementation that accesses an external system.

Figure 7–5 Accessing Data from an External System



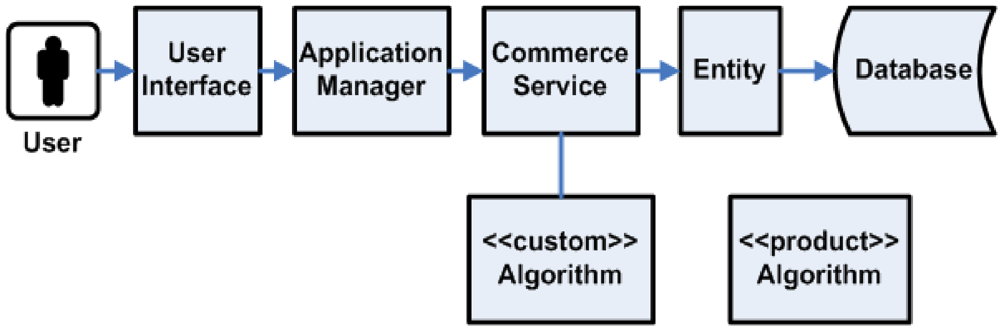
Application impact for this change:

- Deployment Configuration – replacing Commerce Service implementation with custom implementation.

Change an Algorithm Used By a Service

Assuming the UI is held constant, but values such as net totals or other attributes are derived with different calculations, it is advantageous to replace simply the algorithm in question, as the logic flow through the current service does not change.

Figure 7–6 Application Layers



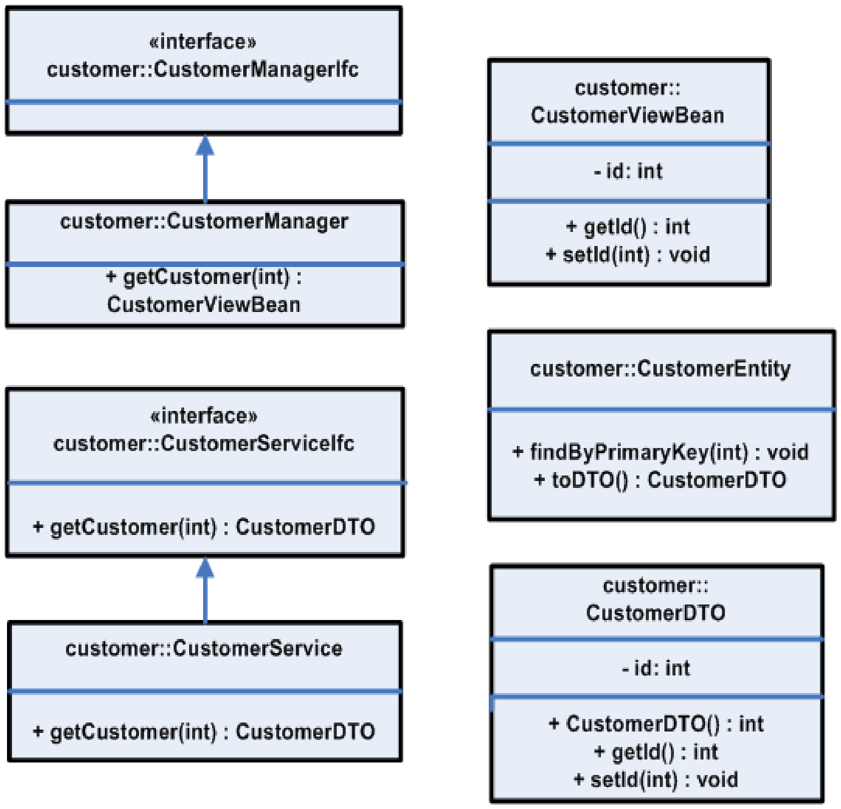
Application impact for this change:

- Algorithm
- Application Configuration

Extension Strategies

Refer to the following diagram as a subset of classes for comparison purposes.

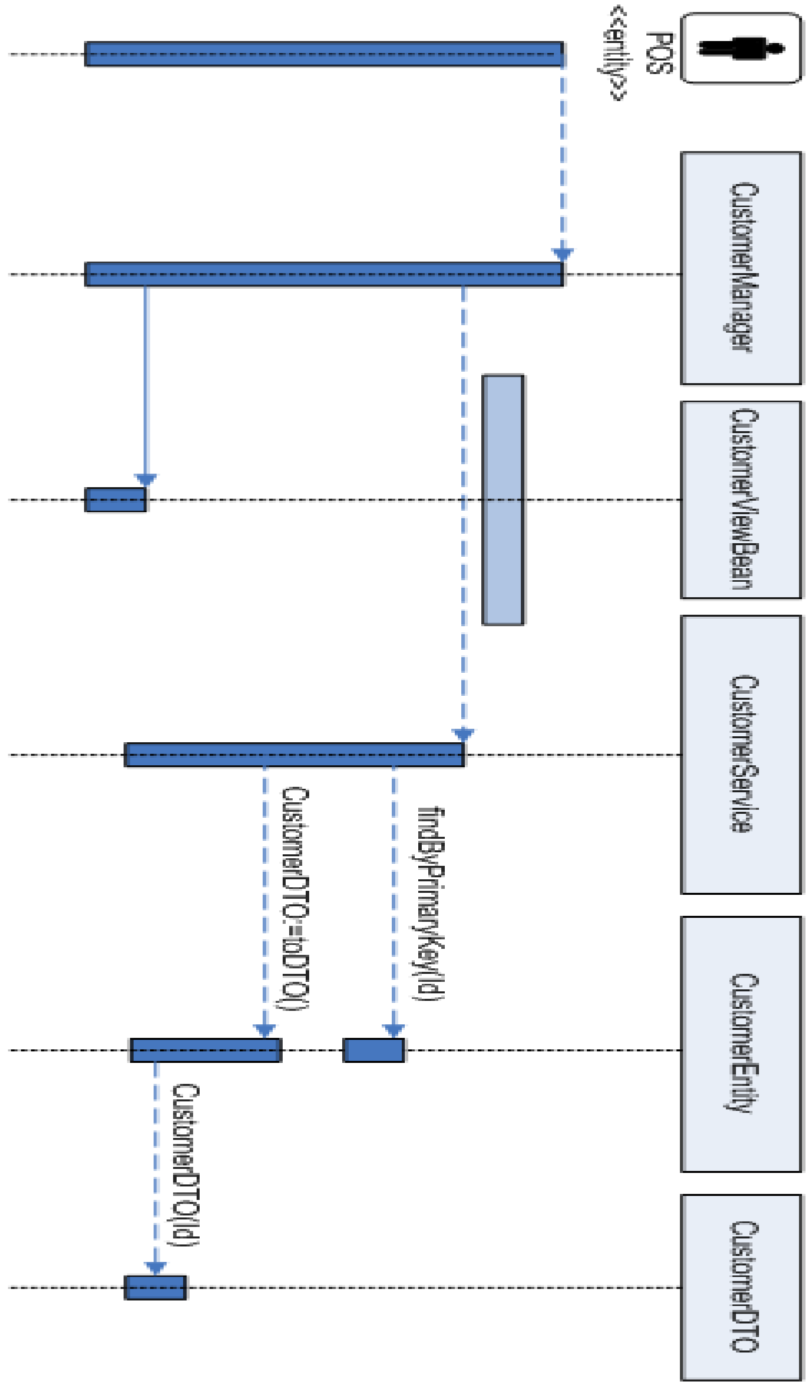
Figure 7–7 Sample Classes for Extension


«interface»
customer::CustomerManagerIfc
customer::CustomerManager
+ getCustomer(int) : CustomerViewBean
«interface»
customer::CustomerServiceIfc
+ getCustomer(int) : CustomerDTO
customer::CustomerService
+ getCustomer(int) : CustomerDTO
customer:: CustomerViewBean
- id: int
+ getId() : int
+ setId(int) : void
customer::CustomerEntity
+ findByPrimaryKey(int) : void
+ toDTO() : CustomerDTO
customer:: CustomerDTO
- id: int
+ CustomerDTO() : int
+ getId() : int
+ setId(int) : void

Extension with Inheritance

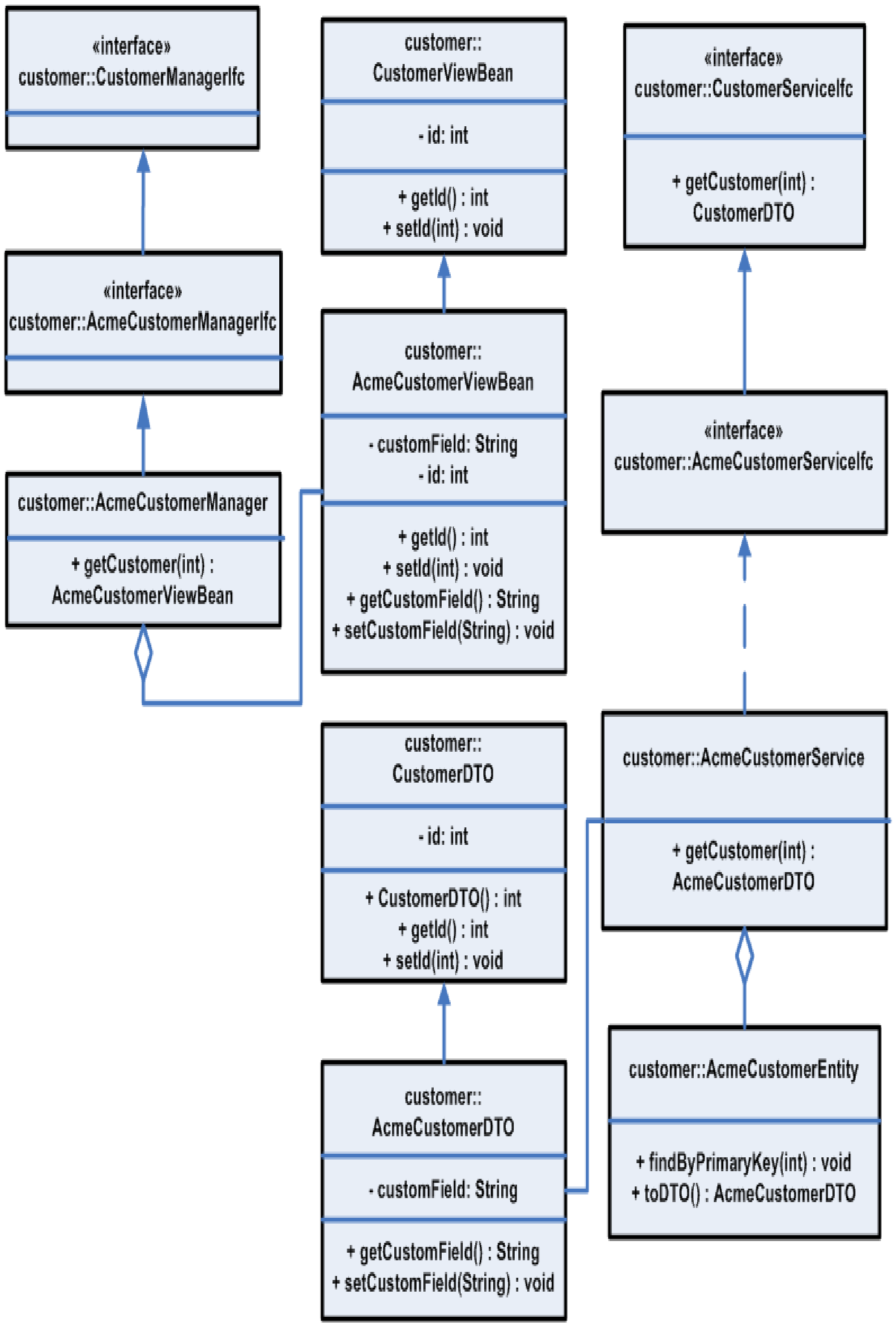
This strategy involves changing the interfaces of the service itself, perhaps to include a new finder strategy or data items unique to a particular implementation. For instance, if the customer information contained in base product does not contain data relevant to the implementation, call it CustomField1.

Figure 7–8 Extension with Inheritance



All of the product code would be extended (the service interface, the implementation, the DTO and view beans utilized by the service, the UI layers and the application manager interface and implementation) to handle access to the new field.

Figure 7–9 Extension with Inheritance: Class Diagram



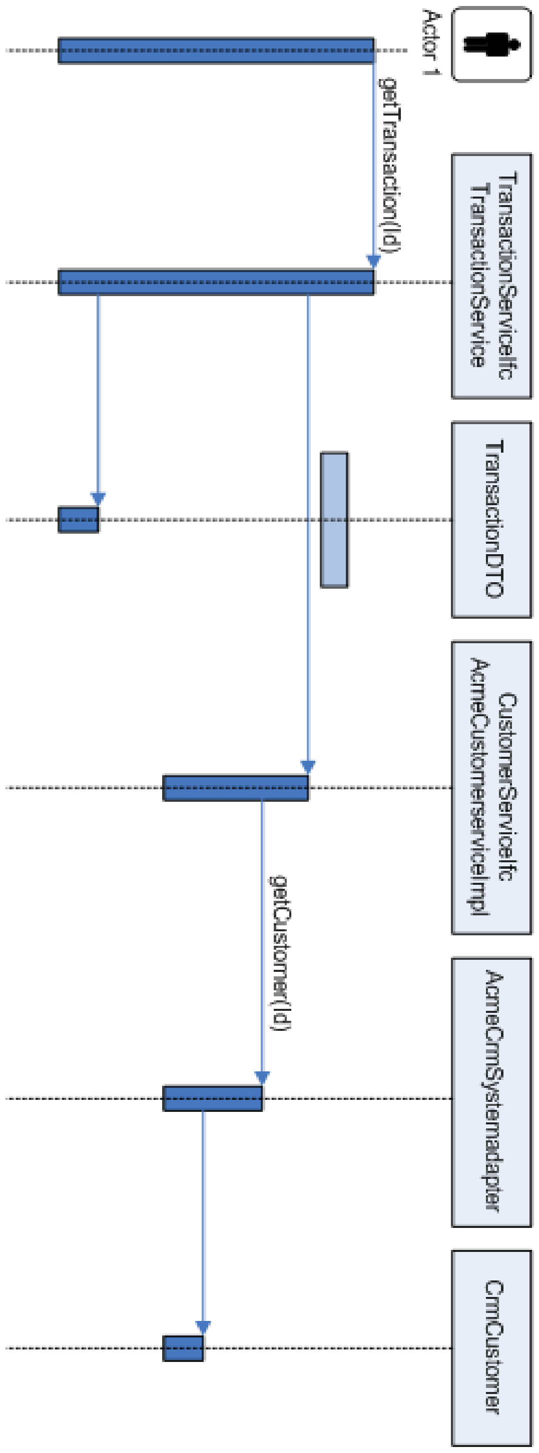
Replacement of Implementation

This strategy involves keeping the existing product interfaces to the service intact, but utilizing a new implementation. This strategy is suggested for when the entire persistence layer for a particular service is changed or delegated to an existing system.

The following diagram demonstrates the replacement of the product Customer Service implementation with an adapter that delegates to an existing CRM solution (the system of record for customer information for the retailer).

This provides access to the data from the existing services that depend on the service interface.

Figure 7–10 Replacement of Implementation



Service Extension with Composition

This method is preferred adding features and data to the base product configuration. This is done with Composition, instead of inheritance.

For specific instances when you need more information from a service that the base product provides, and you wish to control application behavior in the service layer, it is suggested to use this extension strategy. The composition approach to code reuse provides stronger encapsulation than inheritance. Using this method keeps explicit reference to the extended data/operations in the code that needs this information. Also, the new service contains rather than extends the base product. This allows for less coupling of the custom extension to the implementation of the base product.

Figure 7–11 Extension with Composition: Class Diagram

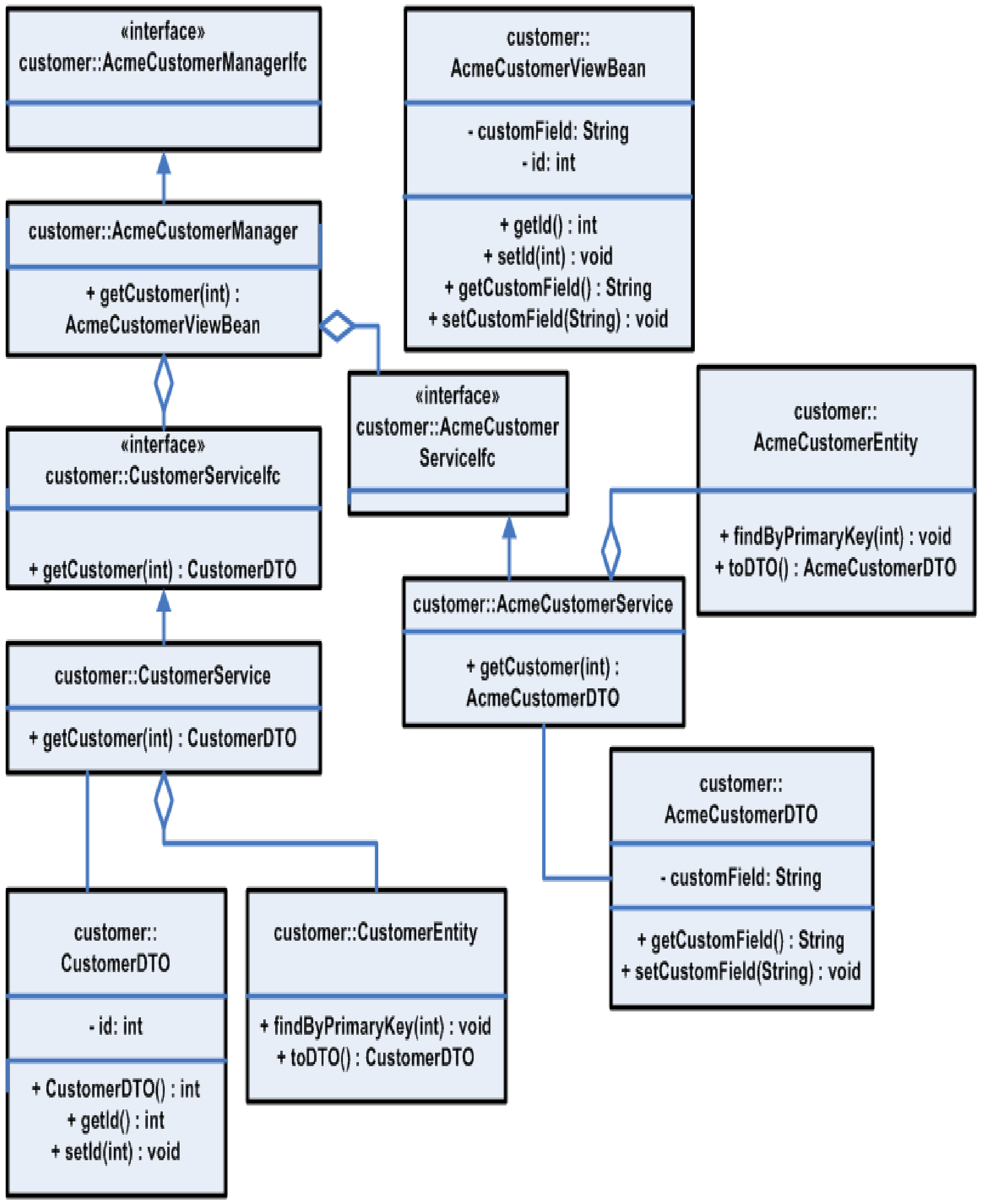
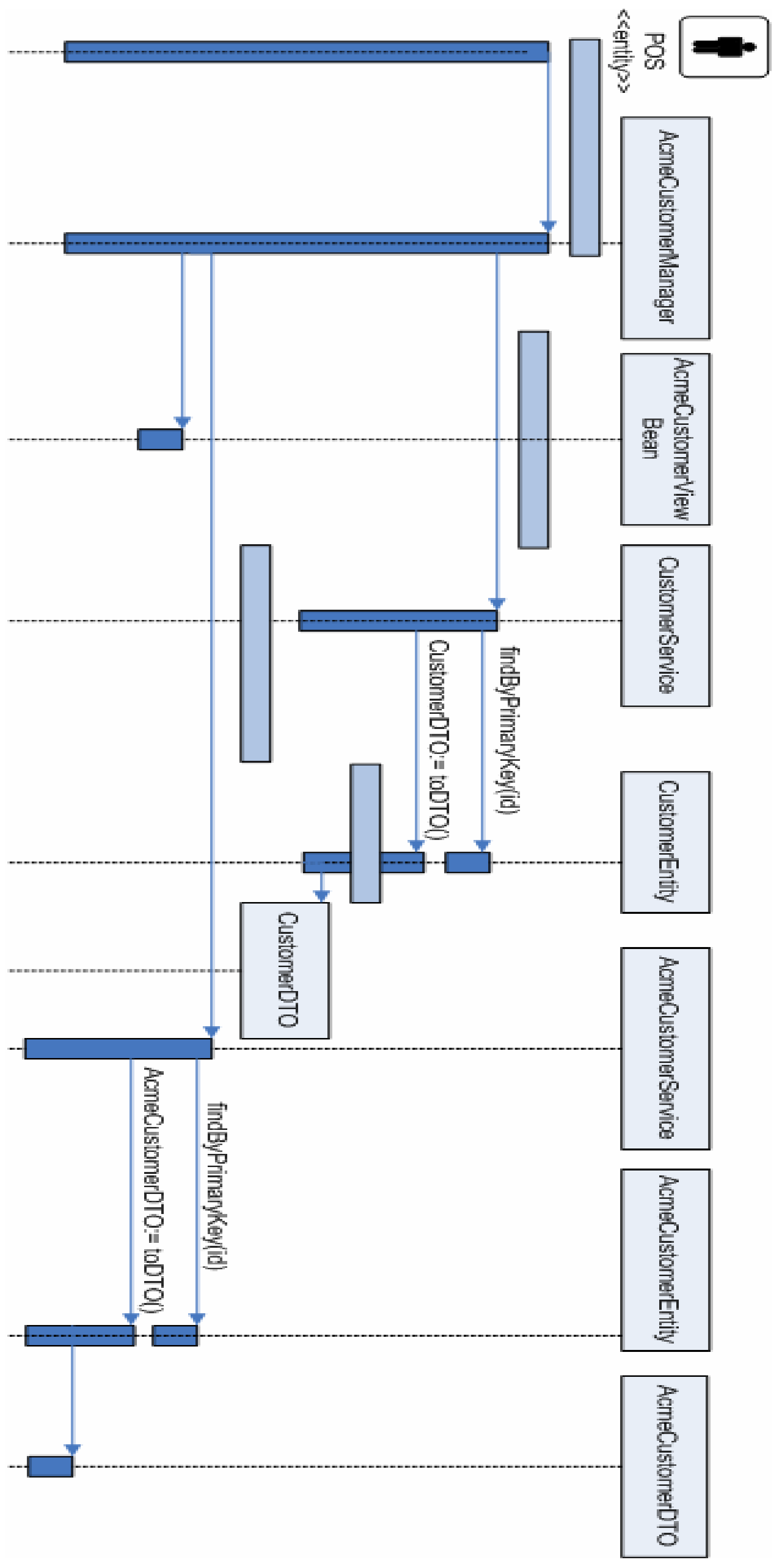

«interface»
customer::AcmeCustomerManagerIfc
customer::AcmeCustomerViewBean
- customField: String
- id: int
+ getId() : int
+ setId(int) : void
+ getCustomField() : String
+ setCustomField(String) : void
customer::AcmeCustomerManager
+ getCustomer(int) : AcmeCustomerViewBean
«interface»
customer::AcmeCustomerServiceIfc
customer::AcmeCustomerEntity
+ findByPrimaryKey(int) : void
+ toDTO() : AcmeCustomerDTO
«interface»
customer::CustomerServiceIfc
+ getCustomer(int) : CustomerDTO
customer::AcmeCustomerService
+ getCustomer(int) : AcmeCustomerDTO
customer::CustomerService
+ getCustomer(int) : CustomerDTO
customer::AcmeCustomerDTO
- customField: String
+ getCustomField() : String
+ setCustomField(String) : void
customer::CustomerDTO
- id: int
+ CustomerDTO() : int
+ getId() : int
+ setId(int) : void
customer::CustomerEntity
+ findByPrimaryKey(int) : void
+ toDTO() : CustomerDTO

Figure 7–12   Extension Composition



Data Extension Through Composition

This strategy describes having the entity layer take responsibility for mapping extra fields to the database by aggregating the custom information and passing it through the service layer. This approach assumes that the extra data is presented to the user of the system and persisted to the database, but is not involved in any service layer business logic.

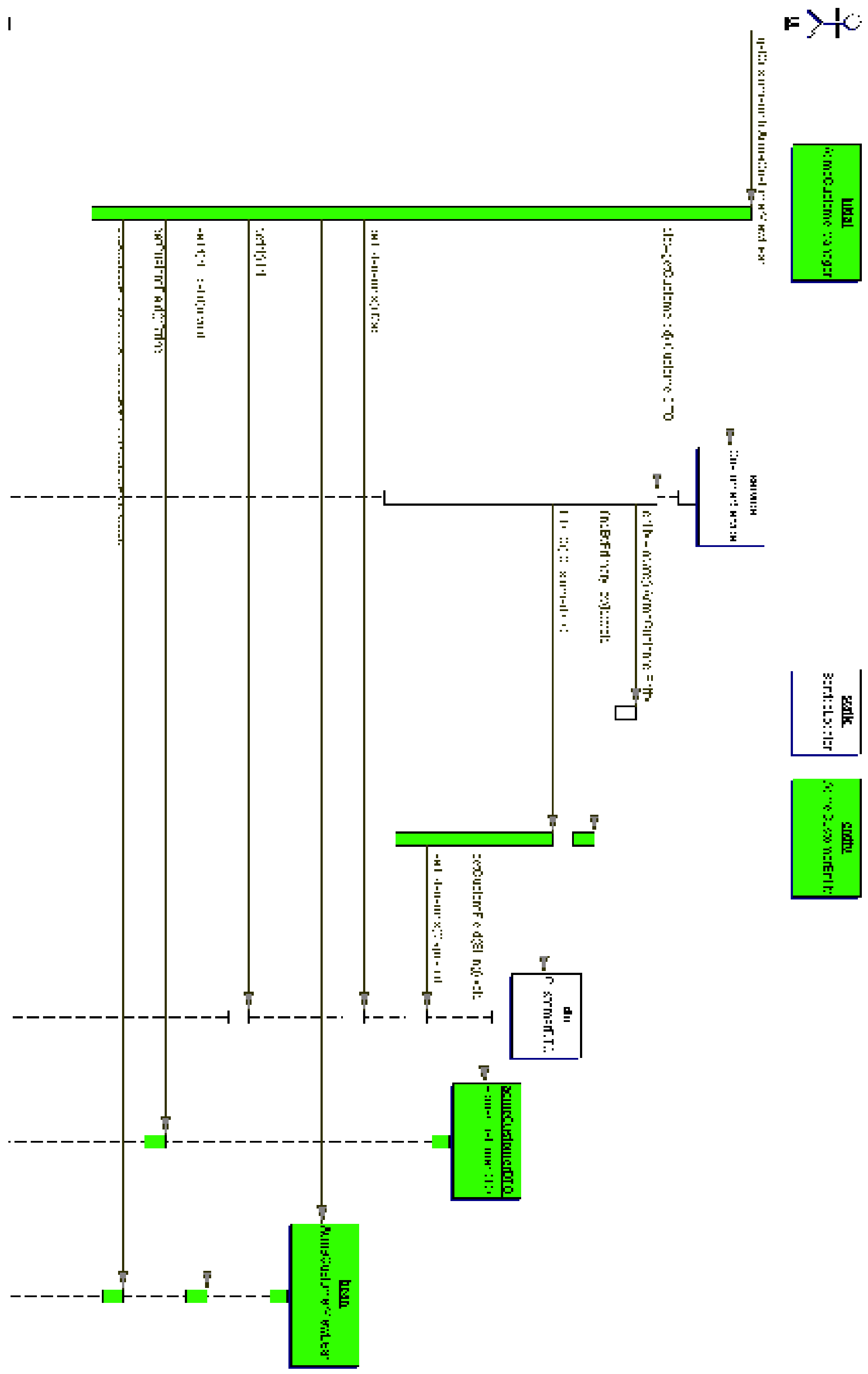
This scenario alters the UI layer (JSP/Action/ViewBean) and adds a new ApplicationManager method to call assemble the ViewBean from the extensible DTO provided by the replaced Entity bean.

Slight modifications to the Service session bean might be necessary to support the toDTO() and fromDTO (ExtensibleDTOIfc dto) methods on the Entity bean, depending on base product support of extensions on the particular entity bean.

1. Create the new ApplicationManager session facade.
2. Create the new ViewBeans required of the UI.
3. Create a new Entity bean that references the original data to construct a base product DTO that additionally contains the custom data using the extensible DTO pattern.
4. Create a new DTO based on the extensible DTO pattern.
5. Create new JSP pages to reference the additional data.
6. Change the deployment descriptors that describe which implementation to use for a particular Entity bean.
7. Change the new Struts configuration and Action classes that reference the customized Application Manager Session facade.
8. If necessary, change the Commerce Service Session facade to give control of the toDTO and fromDTO methods to the Entity bean and do not assemble or disassemble the DTO in this layer, as it does not give a good plug point for the Extensible DTOs.

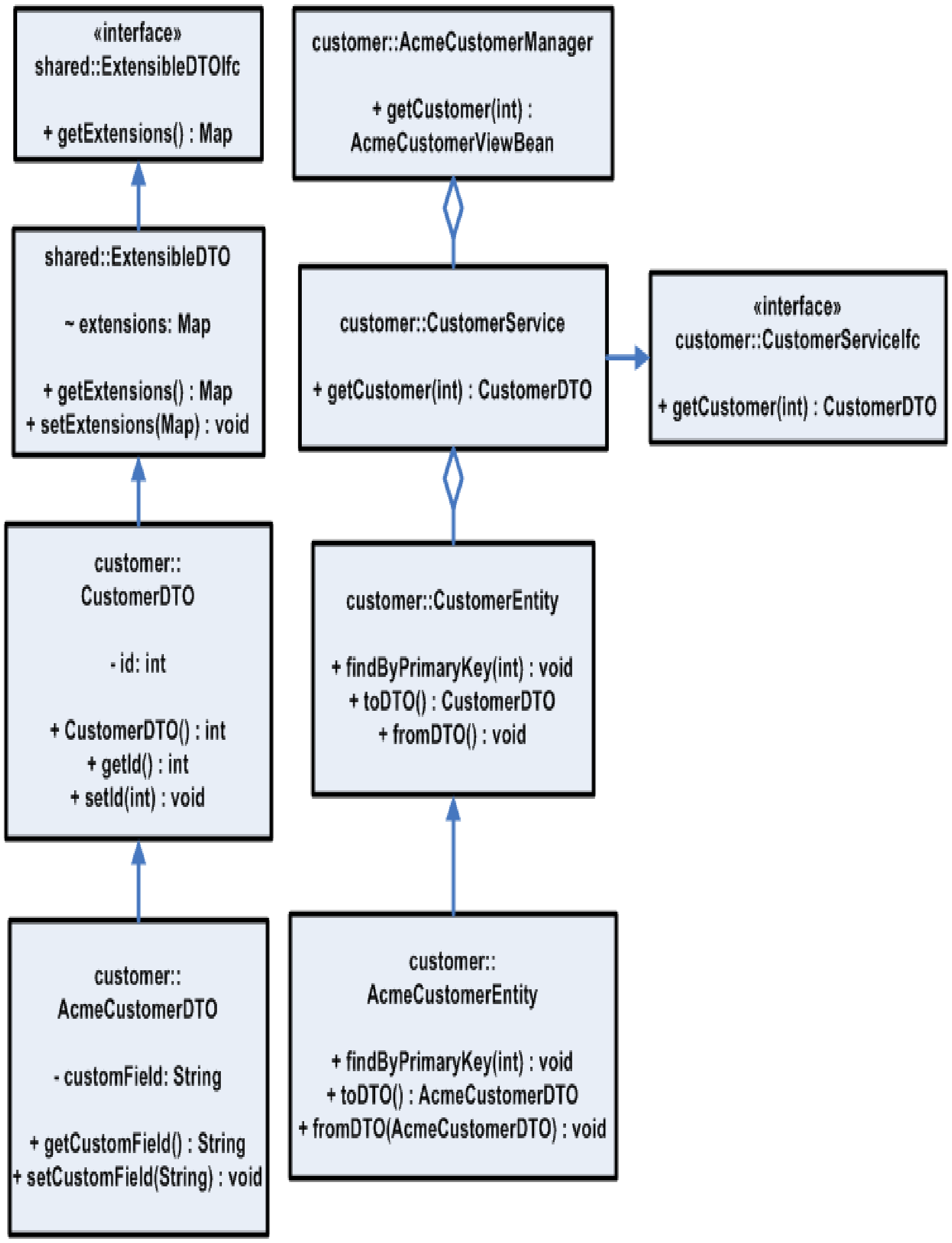
The following diagram describes the life cycle of the data throughout the request.

Figure 7–13 Data Extension Through Composition



The following class diagram describes the various classes created.

Figure 7–14 Data Extension Through Composition: Class Diagram


«interface»
shared::ExtensibleDTOIfc
+ getExtensions() : Map
customer::AcmeCustomerManager
+ getCustomer(int) : AcmeCustomerViewBean
shared::ExtensibleDTO
~ extensions: Map
+ getExtensions() : Map
+ setExtensions(Map) : void
customer::CustomerService
+ getCustomer(int) : CustomerDTO
«interface»
customer::CustomerServiceIfc
+ getCustomer(int) : CustomerDTO
customer::
CustomerDTO
- id: int
+ CustomerDTO() : int
+ getId() : int
+ setId(int) : void
customer::CustomerEntity
+ findByPrimaryKey(int) : void
+ toDTO() : CustomerDTO
+ fromDTO() : void
customer::
AcmeCustomerDTO
- customField: String
+ getCustomField() : String
+ setCustomField(String) : void
customer::
AcmeCustomerEntity
+ findByPrimaryKey(int) : void
+ toDTO() : AcmeCustomerDTO
+ fromDTO(AcmeCustomerDTO) : void

8 Application Services

Application Services request information from Commerce Services and returns that information to the Web UI in a format that can be displayed by the Web UI.

Oracle Retail implements application services in the form of application managers. Application managers aggregate services from multiple Commerce Services into a smaller number of interfaces, and correspond generally to a specific portion of the application user interface.

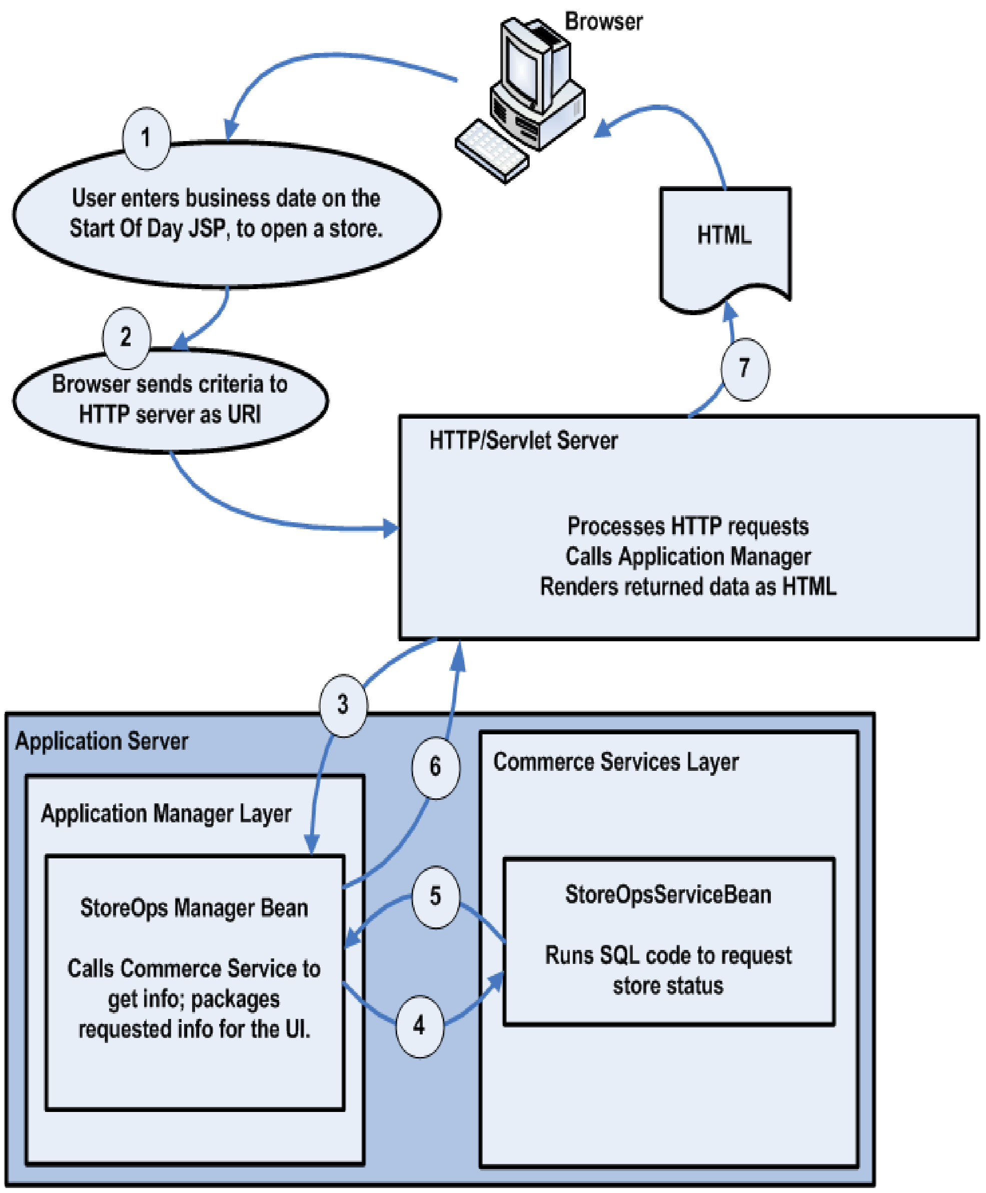
The presence of the Application Services layer offers opportunities for customization that can make your implementation of Back Office more stable across upgrades. This pattern optimizes network traffic, as requests for multiple Commerce Services tend to be funneled through a smaller number of application managers.

These services contain primarily application logic. Business logic should be kept out of these services and instead shared from the Commerce Services tier. In many cases the only function of an Application Service method is to call one or more Commerce Services. Each manager is a facade for one or more Commerce Services. A typical method in the Application Services layer aggregates several method calls from the Commerce Services layer, allowing the real retail business components to remain decoupled from each other.

Application managers are called by Struts Action classes to execute functionality that ultimately derives from Commerce Services. Struts Action classes should not call Commerce Services directly.

The following figure shows how an Application Manager functions within the application.

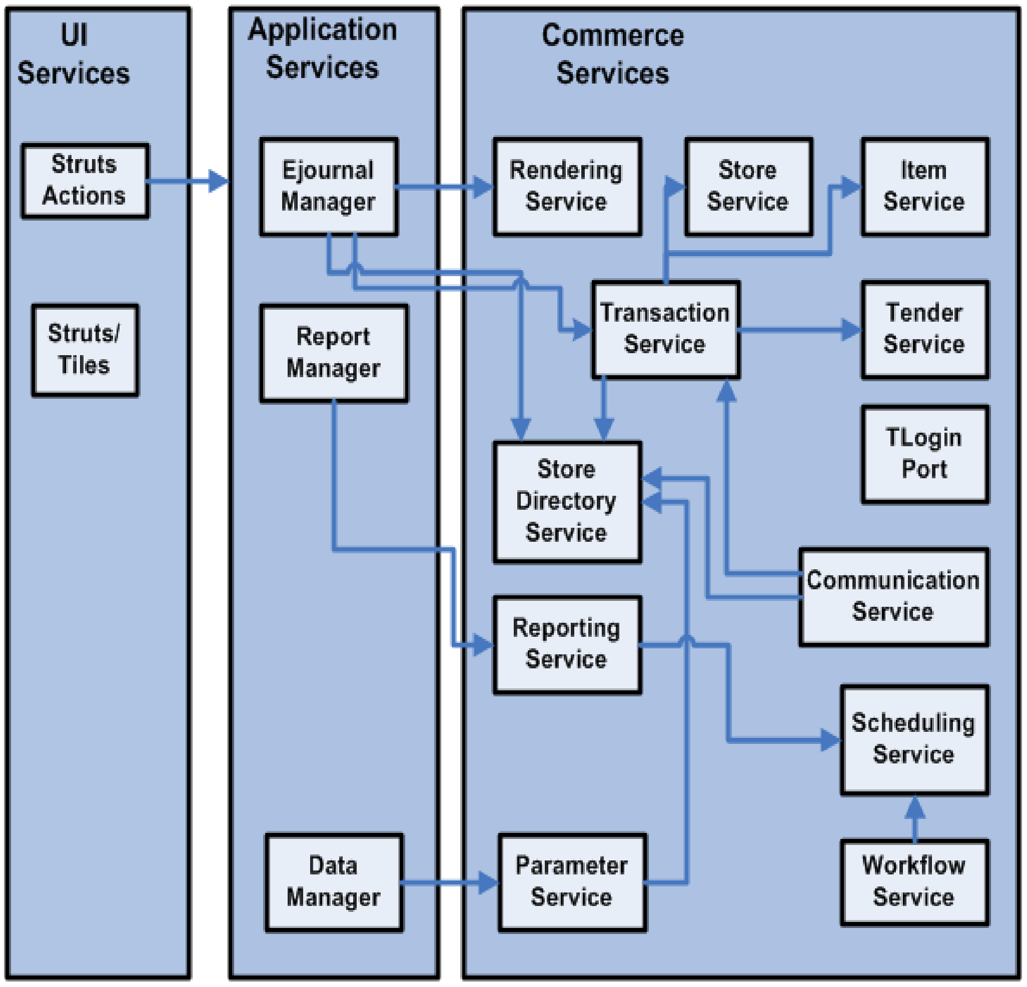
Figure 8–1 Application Manager in Operation


Browser
1
User enters business date on the Start Of Day JSP, to open a store.
2
Browser sends criteria to HTTP server as URI
HTML
7
HTTP/Servlet Server
Processes HTTP requests
Calls Application Manager
Renders returned data as HTML
3
6
Application Server
Application Manager Layer
StoreOps Manager Bean
Calls Commerce Service to get info; packages requested info for the UI.
Commerce Services Layer
StoreOpsServiceBean
Runs SQL code to request store status
5
4

Application Service Architecture

The following diagram shows the relationship between the User Interface, the Application Services, and the Commerce Services.

Figure 8–2 Example Application Service Interactions



Application Manager Mapping

Table 8–1 shows how individual Application Managers map to various parts of the application.

Table 8–1 **Application Manager Mapping**

Tab	Manager	API	Functions
Reports	Report Manager	ReportManagerIfc.java	Displaying, executing, and scheduling the reports
Admin	Task Manager	TaskManagerIfc.java	Scheduling tasks
Employee	Employee Manager	EmployeeManagerIfc.java	Managing security groups, users, and employees
Pricing	Pricing Manager	PricingManagerIfc.java	Managing price changes, promotions, and discount rules
Item	Item Manager	ItemManagerIfc.java	Searching for items
Store Ops	Store Ops Manager	StoreOpsManagerIfc.java	Opening and closing the store, registers, and tills

Extending an Application Manager

The application manager layer provides an opportunity for customizing application behavior without changing the underlying Commerce Services. Some examples of reasons to extend or modify an application manager include:

- To change content that comes from Commerce Services. You can remove data or change how it is handled, formatted, or displayed by changing the logic in the application manager.
- You can provide additional data to your JSPs via the application managers, either by supplying data that comes from existing Commerce Services functions but is not displayed by the default user interface, or by calling new, custom Commerce Services.
- When you add input fields to the user interface, you must make sure that the appropriate application manager knows about those fields and knows how to handle them. If you are extending search criteria, for example, the application manager has to be able to pass those criteria on to the Commerce Service layer.

Creating a New Application Manager

The following steps outline the requirements for making a new application manager:

1. Make new EJB.jar for the application manager.
   - New directory \webapp\<new_app_manager_name>
   - build.xml file to control building with ant
   - sub-directory that contains \classes, \inst, \javadoc, \WEB-INF, \META-INF, \dist, \src, \web and \test directories
   - WEB-INF directory that contains Struts/Tiles config files
   - web directory that contains .jsp files
2. Edit application configuration files.
   - Edit build.xml file for \backoffice\assemble to add your new module to the ejb.webapp.list property list. Depending on content, other lists might also need to be updated.
   - application.xml: add a tag for your EJB to the list of EJBs, as manager_name-admin-ejb.jar.
   - Add the module in the backoffice.env to build this newly added module under build directory.
3. Edit UI files.

Create UI references in Struts configuration files, as described in Chapter 5, "Coding Your First Feature".

Application Manager Reference

All of the managers are stateless session facades which provide functionality in a UI-centric form to be called by Struts Action classes associated with various JSPs. The topics in this section describe the individual application managers.

EJournal Manager

EJournal Manager handles functionality related to the Transaction Tracker tab in the user interface. EJournal Manager enables searches for transactions based on a variety of criteria and combinations of criteria.

The following are dependencies:

- Parameter Service
- Tender Service
- Transaction Service
- Customer Service
- Store Directory
- Reporting Service

Item Manager

Item Manager handles item search functions for the Item tab in Back Office.

The following are dependencies:

- Item Service
- Store Directory

Report Manager

Provides functions for displaying, executing, and scheduling the reports, as well as managing lists of user favorite reports. Supports the application's Reports tab.

The following are dependencies:

- Workflow/Scheduling Service
- Store Directory
- Reporting Service
- ReportGroupTaskExecutionMDB

Store Manager

Provides the ability to read and write information about a store to and from the database. This includes store address and store hierarchy information.

The following are dependencies:

- Workflow/Scheduling Service
- Store Directory
- Employee/User Service

StoreOps Manager

Provides store operations functions. This includes Start of Day, End of Day, and Deposit operations, as well as opening and closing registers and opening and reconciling tills. This manager handles tasks for the Store Ops tab in Back Office.

The following are dependencies:

- StoreOps Service
- Parameters Service
- Currency Service

Task Manager

Handles workflow and displays job information.

The following are dependencies:

- Workflow Service
- File Transfer Service

9 Commerce Services

The topics in this chapter describe each of the available Commerce Services. The Commerce Services in Back Office provide the model component of the MVC pattern; they store the state of data and respond to requests to change that state which come from the controller. The Commerce Services are intended to encapsulate all of the business logic of the application. They are built as session beans, sometimes exposed as Web services, which contain the shared retail business logic. Commerce Services aggregate database tables into objects, combining sets of data into logical groupings. They are organized by business logic categories rather than application functionality. These are services like Transaction, Store Hierarchy, or Parameter, which could be used with any retail-centric application. The Commerce Services talk to the database through a persistence layer of entity beans, described in Chapter 10, "Store Database".

For each service, this chapter includes a description, a listing of the database tables used by the service, plus notes on extending the service and a list of dependencies on other services. The database tables listed are those which are updated by the service directly, excluding any services merely accessed by the service, or which are updated through other services.

Note: For complete and updated database tables for the services listed in this chapter, refer to the *Oracle Retail Strategic Store Solutions Data Model: Relational Integrity Diagrams*.

This chapter covers the following services:

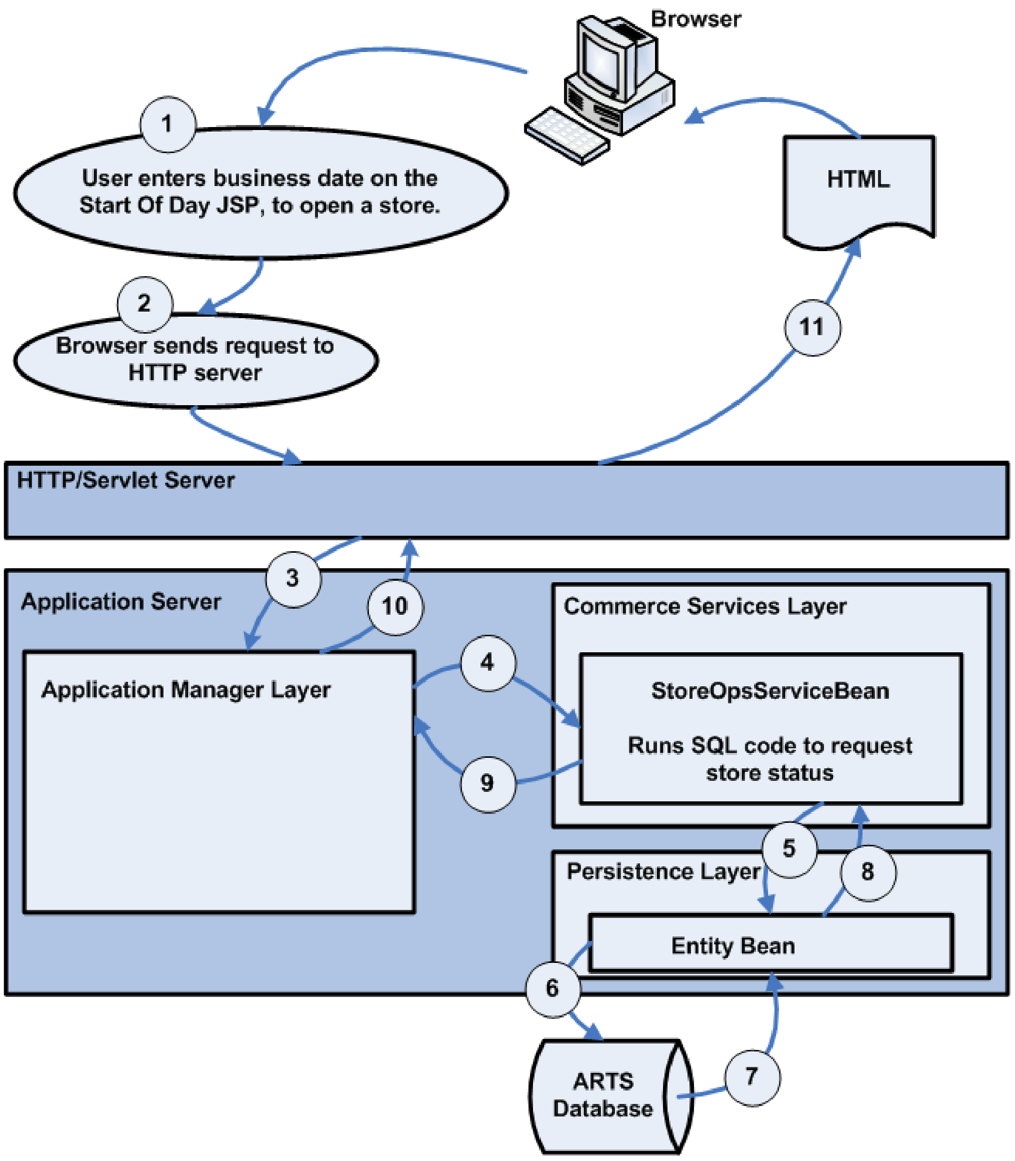
- Calendar Service
- Code List Service
- Currency Service
- Customer Service
- Employee/User Service
- File Transfer Service
- Financial Totals Service
- Item Service
- Parameter Service
- Party Service
- POSlog Import Service
- Post-Processor Service

- Pricing Service
- Reporting Service
- Store Directory Service
- Store Service
- Store Ops Service
- Tax Service
- Time Maintenance Service
- Transaction Service
- Workflow/Scheduling Service

Commerce Services in Operation

The following figure shows how the Commerce Services function within the application.

Figure 9–1 Commerce Services in Operation


Browser
1
User enters business date on the Start Of Day JSP, to open a store.
2
Browser sends request to HTTP server
HTML
11
HTTP/Servlet Server
Application Server
3
10
Application Manager Layer
Commerce Services Layer
4
StoreOpsServiceBean
Runs SQL code to request store status
9
5
8
Persistence Layer
Entity Bean
6
ARTS Database
7

Creating A New Commerce Service

To create a new Commerce Service, use the following basic steps:

1. Make a new EJB.jar with the following components:
   - New directory \COMMERCESERVICES\<*new_service_name*>
   - A build.xml file for ant configurations
   - \classes, \META-INF, \dist, \src and \test directories
2. Edit application configuration files:
   - Edit the build.xml file for \backoffice\assemble to add the module to the ejb.commerceservices.list property list.
   - Edit the application.xml file: add a tag for the EJB to the list of EJBs.

- Add the module in the backoffice.env to build this newly added module under build directory.

3. Edit Application Service and UI files:
   - Update Application Service to call methods in the Commerce Service.
   - Create UI references in Struts configuration files.

Calendar Service

Package of business-calendar-related functionality for reporting.

Database Tables Used

- CA_CLD (Calendar)
- CA_CLD_LV (Calendar Level)
- CA_CLD_PRD (Calendar Period)
- CA_PRD_RP_V4 (Calendar Reporting Period V4)

Interfaces

Access the service through CalendarServiceIfc.java. It provides methods to create and remove calendars as well as various methods to get reporting periods based on various criteria.

Extending This Service

You can extend this service to change the way dates are handled. For example, the default service provides year, month, week, and day as units for reporting. You might want to add quarters to this list. Doing so requires adding code to the service to handle resolving data to the new unit. However, if you wanted to remove one of these units, for example, remove reporting by week, you can do so by changing the database alone.

Dependencies

None.

Tier Relationships

The functionality of this service is the same whether it is used within Central Office or Back Office.

Code List Service

The Code List Service enables web applications to retrieve code lists from various sources:

- Inventory codes
- Advanced Pricing codes
- POS Department codes
- Suspended Transactions codes

Database Tables Used

ID_LU_CD (CodeList)

Interfaces

Access this interface through CodeListServiceIfc.java. The following code sample shows the available methods. It provides methods to obtain reason codes and POS departments based on varying criteria.

Extending This Service

You can add additional codes to the system without extending this service, as it simply retrieves the set of codes that exist.

Dependencies

None.

Tier Relationships

The functionality of this service is the same whether it is used in Central Office or Back Office.

Currency Service

The Currency Service enables you to query for the base local currency setting, from the database. This service also handles addition of currency, including currency in multiple denominations.

Database Tables Used

- CO_CNY (Currency List)
- CO_RT_EXC (Exchange Rates)
- CO_CNY_DNM

Interfaces

Access this interface through CurrencyServiceIfc.java. It provides methods to handle currencies and exchanges rates.

Extending This Service

The default service supports U.S. dollars, Canadian dollars, Mexican pesos, Japanese yen, and British pounds (pound sterling); it can be extended to handle additional currencies, and to enable the addition of multiple currencies to each other, with appropriate handling of exchange rates.

The service can also be extended to connect to an ASP to get exchange rates or other currency information.

Dependencies

None.

Tier Relationships

This service is used only in Back Office.

Customer Service

The Customer Service is used to locate customer information. Typically this information is displayed as additional details to a transaction.

Database Tables Used

PA_CT (Customer)

Interfaces

Access the service through CustomerServiceIfc.java. It provides methods to create, search for, and remove customers.

Extending This Service

In a deployment, you can extend this service by connecting it to a Customer Relationship Management (CRM) application. The service encapsulates the Oracle Retail customer data function so that other portions of the application do not have to change if such a connection is implemented.

Dependencies

Party Service.

Tier Relationships

The functionality of this service is the same whether it is used within Central Office or Back Office.

Employee/User Service

Searches for employees and updates their details, including security permissions.

Database Tables Used

- CO_ACS_GP_RS (GroupResourceAccess)
- CO_GP_WRK (WorkGroup)
- PA_EM (Employee)

Interfaces

Access the service through EmployeeServiceIfc.java. It provides methods to manipulate employees and their permissions.

Extending This Service

You can extend this service by replacing it with a connection to a personnel database or application, such as an LDAP system.

Dependencies

Party Service.

Tier Relationships

The functionality of this service is the same whether it is used within Central Office or Back Office.

File Transfer Service

The File Transfer Service passes arbitrary files from one component of the system to another. It stores the files in the database.

Database Tables Used

- Database Tables Used
- FILE_SET (File Set)
- FILE_SET_ITEM (File Set Item)

Interfaces

Access the service through FileTransferIfc.java. It provides methods to manipulate file sets for transfer.

Extending This Service

If your project has a particularly optimized solution for storing files on a server, you might want to replace this service with your own solution.

Dependencies

Store Directory.

Tier Relationships

The functionality of this service is the same whether it is used within Central Office or Back Office.

Financial Totals Service

This service provides functions for getting financial totals for transactions and till history. Each type of transaction and history has an associated Financial Total calculator to derive the various values in the Financial Totals classes from the particular transaction type.

Database Tables Used

This service does not have persistent storage of its own; it relies on data from other services.

Interfaces

Access the service through FinancialTotalsServiceIfc.java. It provides methods to obtain financial totals for a set of transactions, for a till, for a workstation, or for a store. It also provides a method to summarize financial totals.

Extending This Service

You can extend this service to perform additional financial aggregations. You can add calculators for additional transaction types or alter the existing calculators.

Dependencies

- Item Service
- Currency Service
- Transaction Service

Tier Relationships

The functionality of this service is the same whether it is used in Central Office or Back Office.

Item Service

The Item Service provides item creation, item search, and item record maintenance. This service includes the ability to import item information: a flat file or XML file of item information is imported and can then be processed immediately or scheduled for later upload. The Item Service can take one of three actions on each item listed in an import:

- Add
- Update
- Delete

Database Tables Used

AS_ITM (Item)

AS_ITM_RTL_STR (Retail Store Item)

AS_ITM_STK (Stock Item)

CO_CLN_ITM (Item Collection)

CO_CLR (Item Color)

CO_STYL (Item Style)

CO_SZ (Item Size)

CO_UOM (Item Unit Of Measure)

ID_IDN_PS (POS Identity)

Interfaces

Use ItemServiceIfc.java, which offers methods to get, update, and import items, search for items, and to get specific information about items, such as units of measure, available colors, and available locations.

Extending This Service

You can extend this service to add item information not carried by the default service. You can change this service to delegate to a merchandising system for item classification or item information. Either of these changes can be made by replacing the default service with a new service that adds the new material and references the default service for the rest of its data.

Dependencies

Party Service.

Tier Relationships

The functionality of this service is the same whether it is used within Central Office or Back Office. Central Office does not make use of as many of the available functions as Back Office does; currently, only Back Office can update and add items. However, Central Office can import items.

Parameter Service

This service stores application configuration data and provides methods for creating and distributing that data to store systems.

Database Tables Used

- PARAMETER (Parameter)
- PARAMETER_SET (Parameter Set)
- PARM_EDITOR (Parameter Editor)
- PARM_GROUP (Parameter Group)
- PARM_SET_PARM (Parameter Set Member)
- PARM_SET_TYPE (Parameter Set Type)
- PARM_TYPE (Parameter Type)
- PARM_VAL_PROP (Parameter Possible Values)
- PARM_VALIDATOR (Parameter Validator)
- PARM_VALUE (Parameter Value)
- VAL_PROP_NAME (Validator Property Name)
- VAL_TYPE (Validator Type)

Interfaces

Methods for the service can be found in ParameterServiceIfc.java. It provides methods to work with application parameter sets, their parameters, and their values. It also provides the import facility to load parameters from XML.

Extending This Service

Parameters can be added or removed without changing the Parameter Service, by importing a new master set of parameters.

Dependencies

None.

Tier Relationships

When used in Back Office, this service distributes parameters to Back Office and to Point-of-Service only. When used in Central Office, this service distributes parameters to Central Office, Back Office, and Point-of-Service.

Party Service

The Party Service collects shared party data like addresses, phone numbers and other contact information. Parties are any person or entity that is a party to a transaction, such as an employee, store, or vendor.

Database Tables Used

- LO_ADS (Address)
- PA_CNCT (Contact)
- PA_PRTY (Party)
- PA_PHN (Phone)

Interfaces

The Party Service has no explicit interface file; it is a collection of entities, such as Address and Contact.

Extending This Service

This service can be extended to connect to a third-party contact database to collect its data.

Dependencies

None.

Tier Relationships

The functionality of this service is the same whether it is used within Central Office or Back Office.

POSlog Import Service

Imports POSlog-formatted XML into the database.

Database Tables Used

- AS_DRW_WS (Workstation Drawer)
- AS_ITM_UNK (Unknown Item)
- AS_LY (Layaway)
- AS_TL (Till)
- AS_WS (Workstation
- CA_DY_BSN (Business Day)
- CA_PRD_RP (Reporting Period
- CO_MDFR_CMN (Commission Modifier)
- CO_MDFR_RTL_PRC (Retail Price Modifier)
- CO_MDFR_SLS_RTN_TX (Sale Return Tax Modifier)
- CO_MDFR_TX_EXM (Tax Exemption Modifier)
- DO_CNT_PHY (Physical Count Document)
- DO_CR_STR (Store Credit)
- DO_CRD_GF (Gift Card)
- LE_HST_STR (Store History)
- LE_HST_STR_SF_TND (Store Safe Tender History)
- LE_HST_STR_TND (Store Safe Tender)
- LE_HST_TL (Till History)
- LE_HST_TL_TND (Till Tender History)
- LE_HST_WS (Workstation History)
- LE_HST_WS_TND (Workstation Tender History)
- LE_LTM_MD_TND (Tender Media Line Item)
- LO_ADS (Address)
- LO_EML_ADS (Email Address)
- OR_LTM (Order Line Item)
- OR_LTM_MDFR_RPRC (Order Line Item Retail Price Modifier)
- OR_ORD (Order)
- ORGN_CT (Business Customer)
- PA_CNCT (Contact)
- PA_CT (Customer)
- PA_ID_PRTY_GEN (Party ID Generation)
- PA_PHN (Phone Number)
- PA_PRTY (Party)

- TR_ADS_SLS_RTN (Sale Return Line Item Address)
- TR_CNT_INV (Inventory Count Transaction)
- TR_CTL (Control Transaction)
- TR_FN_ACNT (Financial Accounting Transaction)
- TR_ITM_CPN_T(NCDo upon Tender Line Item)
- TR_LON_TND (Tender Lone Transaction)
- TR_LTM_ALTR (Alteration Line Item)
- TR_LTM_CHK_T(NCD heck Tender Line Item)
- TR_LTM_CR_STR_TND (Store Credit Line Item)
- TR_LTM_CRDB_CRD_TN (Credit Debit Tender Line Item)
- TR_LTM_DSC (Discount Line Item)
- TR_LTM_GF_CF_TND (Gift Certification Tender Line Item)
- TR_LTM_GF_CRD_TND (Gift Card Tender Line Item)
- TR_LTM_PHY_CNT (Physical Count Line Item)
- TR_LTM_PRCH_ORD_TND (Purchase Order Tender Line Item)
- TR_LTM_PYAN (Payment On Account Line Item)
- TR_LTM_RTL_TRN (Retail Transaction Line Item)
- TR_LTM_SLS_RTN (Sale Return Line Item)
- TR_LTM_SLS_RTN_TX (Sale Return Tax Line Item)
- TR_LTM_SND_CHK_TND (Send Check Tender Line Item)
- TR_LTM_TND (Tender Line Item)
- TR_LTM_TND_CHN (Tender Change Line Item)
- TR_LTM_TRV_CHK_TND (Travelers Check Tender Line Item)
- TR_LTM_TX (Tax Line Item)
- TR_PKP_TND (Tender Pickup Transaction)
- TR_RCV_FND (Funds Receipt Transaction)
- TR_RTL (Retail Transaction)
- TR_SLS_PS_NO (POS No Sale Transaction)
- TR_STR_OPN_CL (Store Open Close Transaction)
- TR_TL_OPN_CL (Till Open Close Transaction)
- TR_TRN (Transaction)
- TR_VD_PST (Post Void Transaction)
- TR_WS_OPN_CL (Workstation Open Close Transaction)

Interfaces

The functions of the POSlog Import Service are encapsulated within the Transaction Service; if you need to call for a POSlog Import, do so through the Transaction Service.

Extending This Service

You can extend this service to capture additional custom POSLog elements which are not part of the base ARTS IXRetail XML standard.

Dependencies

None.

Tier Relationships

The functionality of this service is the same whether it is used within Central Office or Back Office.

Post-Processor Service

The Post-Processor Service provides a service interface for processing transactional data after it is received and storing the information in summary tables. Post-processing serves as a performance enhancement for reports.

Database Tables Used

- LE_SMY_EM_TMACV (EmployeeProductivitySummary)
- LE_SMY_EM_SLS (EmployeeSalesSummary)
- LE_SMY_EM_WS_TMACV (EmployeeWorkstationProductivitySummary)
- LE_SMY_DPT_SLS (FlashSalesDepartmentSummary)
- LE_SMY_MRH_SLS (FlashSalesMerchandiseHierarchySummary)
- LE_SMY_FLSH_SLS (FlashSalesSummary)
- LE_SMY_PS_DPT (PosDepartmentSummary)
- LE_SMY_TL_SLS (TillSalesSummary)
- LE_SMY_WS_SLS (WorkstationSalesSummary)

Interfaces

This service offers a simple interface: there is only one method, processTransactions(). Access this service through PostProcessorServiceIfc.java.

Extending This Service

This service is designed to support a variety of post-processors, which can be created as separate objects. You can extend this service by adding additional post-processors.

Dependencies

- Calendar Service
- Financial Totals Service
- Transaction Service

Tier Relationships

This service is used only in Back Office.

Pricing Service

The Pricing service offers functions for requesting pricing rules, modifying the pricing rules, and creating new pricing rules.

Database Tables Used

- MA_PRC_ITM (ItemPriceMaintenance)
- MA_ITM_PRN_PRC_ITM (PermanentPriceChangeItem)
- TR_CHN_PRN_PRC (PermanentPriceChange)
- MA_ITM_TMP_PRC_CHN (TemporaryPriceChangeItem)
- TR_CHN_TMP_PRC (TemporaryPriceChange)
- CO_EL_PRDV_DPT (DepartmentPriceDerivationRuleEligibility)
- CO_EL_PRDV_ITM (ItemPriceDerivationRuleEligibility)
- CO_PRDV_ITM (ItemPriceDerivation)
- CO_EL_MRST_PRDV (MerchandiseStructurePriceDerivationRuleEligibility)
- TR_ITM_MXMH_PRDV (MixAndMatchPriceDerivationItem)
- RU_PRDV (PriceDerivationRule)
- RU_TY_PRDV (PriceDerivationRuleType)
- CO_EV (Event)
- CO_MNT_ITM (ItemMaintenanceEvent)
- CO_EV_MNT (MaintenanceEvent)
- AS_ITM_RTL_STR (Retail Store Item)

Interfaces

Access this interface through PricingServiceIfc.java. It provides methods to work with pricing rules, price changes, and promotions. It also provides the import facility to import pricing data.

Extending This Service

You can extend this service to add additional pricing functions or to draw data from a different source, such as a marketing database that tracks upcoming price promotions.

Dependencies

- Item Service
- Workflow Service

Tier Relationships

This service is used only in Back Office.

Reporting Service

The Reporting Service is a framework for creating and exporting reports, managing users' favorite reports, and maintaining collections for scheduling. This service supports XML/XSL reports.

Export formats include XML, CSV, PDF and RTF.

Database Tables Used

- EXECUTED_REPORT (Executed Report)
- FAVORITE_REPORT (Favorite Report)
- REPORT_CONFIG (Report Configuration)
- REPORT_CONFIG_PARAMETER (Report Configuration Parameters)
- REPORT_CRITERIA (Report Criteria)
- REPORT_GROUP (Report Group)
- REPORT_RECIPIENT (Report Recipient)

Interfaces

The Reporting Service includes methods for report creation and report type. It is contained within ReportingServiceIfc.java. It provides methods to determine what reports are configured for the application as well as methods to invoke those reports. It also provides for the working with and scheduling favorite reports for later execution.

Extending This Service

The Reporting Service can be easily extended to support new XSL Reports. Report definitions are database driven. The report definitions contain a name, report implementation, report parameters, and report types.

Dependencies

- Workflow/Scheduling Service
- Store Service
- Store Ops Service
- Calendar Service

Tier Relationships

The functionality of this service is the same whether it is used within Central Office or Back Office.

Store Directory Service

This service provides access to the directory of stores in the enterprise.

Database Tables Used

- CO_STRGP_FNC (RetailStoreGroupFunction)
- ST_ASCTN_STRGP_STR (AssociatedRetailStoreStoreGroup)
- CO_STRGP_LV (RetailStoreGroupLevel)
- CO_STRGP (RetailStoreGroup)
- ST_ASCTN_STRGP (AssociatedRetailStoreGroup)

Interfaces

Use StoreDirectoryIfc.java, which offers more than 20 methods. These include methods for getting paths to stores, the current store's node in the hierarchy, or a set of stores based on some set of search criteria. It provides methods to work with the store hierarchy and store groups.

Extending This Service

You can replace this service with a connection to an existing database of stores, if your enterprise already maintains this information in another form.

Dependencies

Parameter Service.

Tier Relationships

The functionality of this service is the same whether it is used within Central Office or Back Office.

Store Service

Look up and maintain store attributes, store hierarchy information and store history.

Database Tables Used

- CA_DY_BSN (Business Day)
- CA_PRD_RP (Reporting Period)
- CO_STRGP (Store Group)
- CO_STRGP_FNC (Store Group Function)
- CO_STRGP_LV (Store Group Level)
- EMPLOYEE_HIERARCHY_ASSN (Employee Hierarchy Association)
- LE_HST_STR (Store History)
- LE_HST_STR_SF_TND (Store Safe Tender History)
- LE_TND_STR_SF (Store Safe Tender)
- PA_STR_RT (Retail Store)

- ST_ASCTN_STRGP (Associated Retail Store Group)
- ST_ASCTN_STRGP_STR (Associated Retail Store Group Store)

Interfaces

Use StoreServiceIfc.java. It provides methods to obtain store and workstation information. It also provides a facility to update workstation parameters for a store.

Extending This Service

If your enterprise needs additional store information not carried by the default service, you can extend this service to include the new data.

Dependencies

Parameter Service.

Tier Relationships

The functionality of this service is the same whether it is used within Central Office or Back Office.

Store Ops Service

This service provides functions for opening and closing the store, as well as other store operations.

Database Tables Used

The service depends on other services for its data and does not access persistent storage directly.

Interfaces

Use StoreOpsServiceIfc.java, which includes methods for opening and closing the store plus querying whether the store is currently open, opening and closing a specific workstation, and handling tills.

Extending This Service

You can extend or modify this service to change how stores are opened, closed, or reconciled.

Dependencies

Parameter Service.

Tier Relationships

This service is used only in Back Office.

Tax Service

The Tax service enables tax information to be imported from a tax file.

Database Tables Used

- CO_GP_TX_ITM (TaxableGroup)
- PA_ATHY_TX_PSTL (TaxAuthorityPostalCode(Deprecate))
- PA_ATHY_TX (TaxAuthority)
- RU_TX_GP (TaxGroupRule)
- RU_TX_RT (TaxRateRule)

Interfaces

Access this interface through TaxServiceIfc.java. There is only one method, importTaxFile().

Extending This Service

You can replace this service with one to import tax information from a different source. If you want this service to perform other functions when importing a tax file, you can wrap this service with one of your own creation, calling this service to perform the tax file import.

Dependencies

Party Service.

Tier Relationships

This service is used only in Back Office.

Time Maintenance Service

The Time Maintenance Service provides an interface to functions which manage employee work time data. This includes clocking in and clocking out, editing, creating, and confirming employee time.

Database Tables Used

- ADT_LOG (AuditLog)
- CO_CONF_EM_TM_ENR (EmployeeConfirmedTimeEntry)
- CO_EM_TM_ENR (EmployeeTimeEntry)
- CA_WRK_WK (WorkWeekConfirm)

Interfaces

Access this interface through TimeMaintenanceServiceIfc.java. It provides methods to work with time entries and to validate and confirm those entries on a weekly basis. Facilities to audit, summarize, and edit employee hours are also provided.

Extending This Service

You can extend this service to add additional time maintenance functions, or to connect to an application other than Back Office to supply time-tracking information.

Dependencies

Calendar Service.

Tier Relationships

This service is expected to be used only with Back Office, although its functionality works within Back Office and Central Office.

Transaction Service

Searches for transactions and E-journals based on transaction, customer or item information.

Database Tables Used

- AS_ITM (Item)
- AS_ITM_STK (Stock Item)
- CO_MDFR_CMN (Commission Modifier)
- CO_MDFR_RTL_PRC (Retail Price Modifier)
- ID_IDN_PS (POS Identity)
- JL_ENR (Journal Entry)
- LE_LTM_MD_TND (Tender Media Line Item)
- LO_ADS (Address)
- PA_CNCT (Contact)
- PA_CT (Customer)
- TR_CTL (Control Transaction)
- TR_LON_TND (Tender Lone Transaction)
- TR_LTM_CHK_TND (Check Tender Line Item)
- TR_LTM_CRDB_CRD_TN (Credit Debit Tender Line Item)
- TR_LTM_RTL_TRN (Retail Transaction Line Item)
- TR_LTM_SLS_RTN (Sale Return Line Item)
- TR_LTM_TND (Tender Line Item)
- TR_PKP_TND (Tender Pickup Transaction)
- TR_RTL (Retail Transaction)
- TR_SLS_PS_NO (POS No Sale Transaction)
- TR_STR_OPN_CL (Store Open Close Transaction)
- TR_TL_OPN_CL (Till Open Close Transaction)
- TR_TRN (Transaction)

- TR_VD_PST (Post Void Transaction)
- TR_WS_OPN_CL (Workstation Open Close Transaction)

Interfaces

The file TransactionServiceIfc.java contains a number of methods for accessing the service. These include transaction creation, search result, and others.

Extending This Service

You must extend this service if you intend to add new transaction types or new ways of searching for transactions.

Dependencies

- Parameter Service
- Customer Service
- Item Service
- Store Service

Tier Relationships

The functionality of this service is the same whether it is used within Central Office or Back Office. Although full functionality is available to both applications, Central Office tends to import transactions while Back Office tends to export them, due to the intended use of the applications.

Workflow/Scheduling Service

Create and edit tasks, schedule tasks, track task approval. Even for tasks which should be scheduled immediately, the Workflow/Scheduling Service provides task tracking features.

Database Tables Used

- CO_EVT_MSG (Job Event Messages)
- FILE_SET (File Set)
- FILE_SET_ITEM (File Set Item)
- SCHEDULE (Schedule)
- TASK (Task)
- TASK_DESTINATION_STATUS (Task Destination Status)
- TASK_HISTORY (Task History)
- TASK_NOTIFICATION_RECIPIENT (Task Notification Recipient)
- TASK_REVIEW (Task Review)
- WORKFLOW_CONFIGURATION (Workflow Configuration)

Interfaces

The methods for this service are defined in WorkflowServiceIfc.java. They include methods for task creation, notification, task destination, and more.

Extending This Service

Extend this service by adding new task types. You must add the new task type to the workflow configuration table, add a map to execute the task, and add a user interface to enable the task type to be created.

Dependencies

- Parameter Service
- Store Service

Tier Relationships

This service is the same whether it is used within Central Office or Back Office; either application calls the service to schedule tasks.

10 Store Database

Oracle Retail Back Office uses an ARTS-compliant database. Data is stored and retrieved by entity beans in a bean-managed persistence pattern, so the system makes database calls from the entity bean code.

A single entity bean exists for each database table, and handles reads and writes for that table. Each entity bean contains the necessary methods to create, load, store, and remove its object type.

The Back Office application writes data to the Store database, a repository for transaction information for a single store.

Related Documentation

Table 10–1 lists related sources that provide specific information about the database for use when developing code.

Table 10–1 Related Documentation

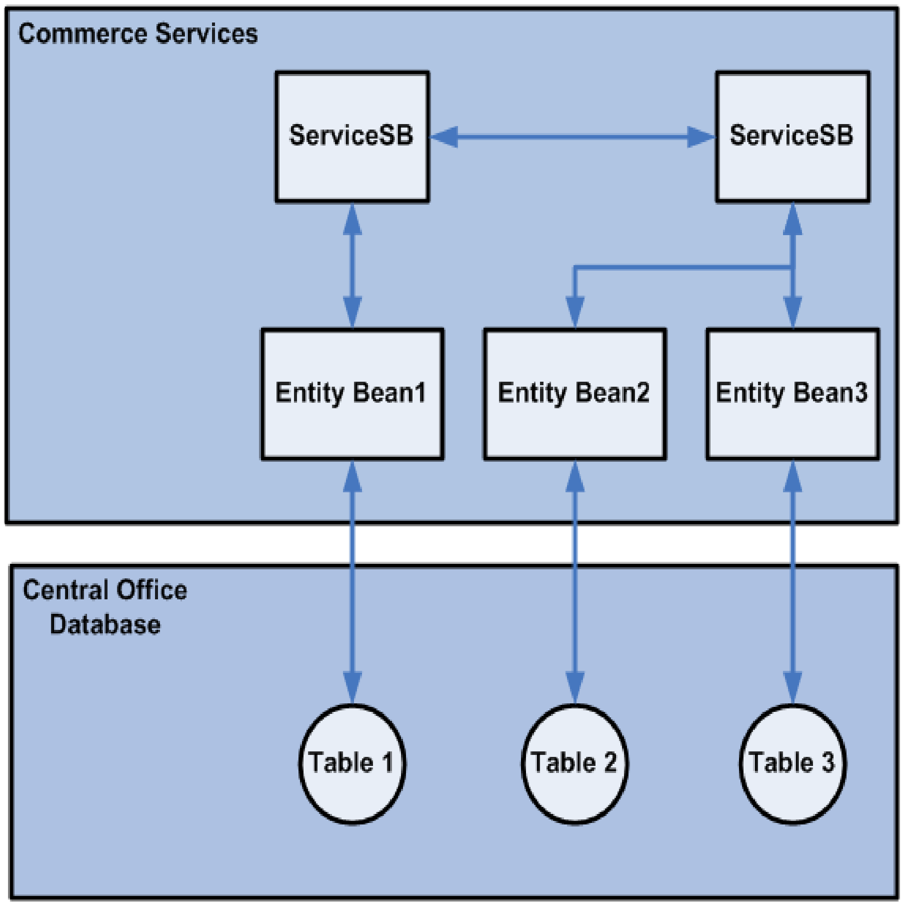
Source	Description
ARTS Database Standard	See `http://www.nrf-arts.org/` for a description of the ARTS database standard.
Data Dictionary	Contains table and column definitions for the database used to store Back Office data. See the _resources directory provided with your Back Office documentation.
Database Diagrams	See the docs .zip file for diagrams which show the relationships between various tables in the database schema.

Database/System Interface

As described in Chapter 2, "Technical Architecture", a persistence layer of entity beans represents the database tables to the rest of the system. One bean represents each table.

The following figure illustrates these relationships.

Figure 10–1 Commerce Services, Entity Beans, and Database Tables



Each commerce service communicates with one or more entity beans, and each entity bean communicates with one database table. Although there are exceptions, in general only one commerce service communicates with an entity bean; other services request the information from the relevant service rather than talking directly to the entity bean. For example, if the Customer Service needs information provided by the Item Bean, it makes a request to the Item Service.

ARTS Compliance

When new code is added or features are added, modified, or extended, database plans should be evaluated to ensure that new data items fit the ARTS schema. Complying with the standards increases the likelihood that extensions can migrate into the product codebase and improves code reuse and interoperability with other applications.

Note: Because the ARTS standard continues to evolve, older code might contain deviations from the standard or might be compliant only with an earlier version of the ARTS standard. Oracle Retail continues to evaluate ARTS compliance with each release of its software.

Bean-Managed Persistence in the Database

In general, the system uses standard J2EE bean-managed persistence techniques to persist data to the Store database. Each of the entity beans that stores data requires JDBC code in standard ejbLoad, ejbStore, ejbCreate, and ejbRemove classes. However, there are some differences worth noting:

- All SQL references are handled as constant fields in an interface.
- Session and entity beans extend an EnterpriseBeanAdapter class. Special extensions for session and entity beans exist. These contain common code for logging and a reference to the Oracle Retail DBUtils class (which provides facilities for opening and closing data source connections, among other resources).

Example 10–1 ItemPriceDerivationBean.java: ejbStore Method

```
public void ejbStore() throws EJBException
    {
        ItemPriceDerivationPK key = (ItemPriceDerivationPK)
getEntityContext().getPrimaryKey();
        getLogger().debug("store");
        PreparedStatement ps = null;
        Connection conn = null;
        if (isModified())
        {
            getLogger().debug("isModified");
            try
            {
                conn = getDBUtils().getConnection();
                ps = conn.prepareStatement(ItemPriceDerivationSQLIfc.STORE_SQL);
                int n = 1;
                ps.setBigDecimal(n++,getReductionAmount().toBigDecimal());
                ps.setBigDecimal(n++,getDiscountPricePoint().toBigDecimal());
                getDBUtils().preparedStatementSetDate(ps, n++,
getRecordCreationTimestamp());
                ps.setBigDecimal(n++,getReductionPercent().toBigDecimal());
                getDBUtils().preparedStatementSetDate(ps, n++,
getRecordLastModifiedTimestamp());
                ps.setInt(n++, key.getPriceDerivationRuleID());
                ps.setString(n++, key.getStoreID());
                if (ps.executeUpdate() != 1)
                {
                    throw new EJBException("Error storing (" +
getEntityContext().getPrimaryKey() + ")");
                }
                setModified(false);
            }
            catch (SQLException ex)
            {
                getLogger().error(ex);
                throw new EJBException(ex);
            }
```

```
                catch (Exception ex)
                {
                    getLogger().error(ex);
                    throw new EJBException(ex);
                }           finally
                {
                    getDBUtils().close(conn, ps, null);
                }
            }
        }
```

A

Appendix: Back Office Data Purge

Data purging is based upon logical sets of data. Logical sets of data can be contained in multiple tables. An example of a logical set of data is all the records associated to a particular Retail Transaction.

A purge of a logical set is not considered complete until all relevant rows of data are deleted.

Data purging is based upon a data-retention schedule whereupon all data existing prior to the computed date are purged. The data within this timeframe must meet constraints as required. For example, if a customer wants to retain the last 180 days worth of retail transaction data, then the integer 180 should be passed into the purge retail transaction routine and the system will purge COMPLETED transactions more than 180 days old.

The stored procedures read the absolute value of a negative integer. For example, a value of **-30** passed into the stored procedures will be read as **30**, and the data will be retained for 30 days.

If no value is passed into the stored procedures, then the default value is used. The default value is 30.

The number of data retention days is passed into the stored procedures. The constraints are built into the stored procedures and are therefore not parameterized.

A logical set purge will succeed even if data is not found in an expected table.

The Financial History and Financial Summary data purge scripts do not address the issue of the weekly sum of daily totals that will no longer match weekly totals. For example, if the purge occurs on a Wednesday, the sum of the daily totals from Wednesday through Saturday will not match the weekly total that was based upon a Sunday through Saturday timeframe.

Caution: Passing in a **zero** (**0**) as a parameter to the purge transaction routines will result in the deletion of all completed transactional data. Oracle is not responsible for loss or damage of any sort that might incur from passing in zero as a parameter.

The customer is fully responsible for the database configuration. Oracle assumes the purge routines will operate within the confines of the database configuration, such as the size of the rollback segments and other such parameters that might affect the functioning of the purge routines.

Invoking Stored Procedures

The following are examples of how to invoke stored procedures.

Note: It is assumed that the user calling the stored procedures has the necessary privileges to invoke these procedures.

Example A–1 Invoking The Stored Procedures -- SQL Plus Method 1

```
SQL> execute <procedure name (parameters)>;

Example:

execute Purge_Fn_Smy(90);
```

Example A–2 Invoking The Stored Procedures -- SQL Plus Method 2

```
SQL> BEGIN
SQL> <procedure name (parameters)>;
SQL> END;

Example:

SQL> BEGIN
SQL> Purge_Fn_Smy(90);
SQL> END;
```

You can choose to create a script file that contains these commands and have a scheduler execute the script on a nightly basis. To do this, you must be logged into the database.

The scheduler must be able to log in to the database to be able to run the scripts, or the log in must be the first line in the script. If the script contains the login, ensure the acount password will not be visible to unauthorized users.

Calls to Invoke Stored Procedures

Table A–1 contains the calls to use to invoke stored procedures.

Table A–1 **Stored Procedure Calls**

Subject Area	Procedure Call
Retail Transactions	Purge_trl_trn (<number of retention days>)
Control Transactions	Purge_ctl_trn (<number of retention days>)
Financial Accounting Transactions	Purge_fn_trn (<number of retention days>)
Orders	Purge_ord (<number of retention days>)
Layaways	Purge_ly (<number of retention days>)
Financial Histories	Purge_Fn_Hst (<number of retention days>)
Financial Summaries	Purge_Fn_Smy (<number of retention days>)
Advanced Pricing	Purge_prdv (<number of retention days>)
eJournal	Purge_ejrl (<number of retention days>)

Restricting Access To Data Purge Scripts

Consider the following when restricting access to data purge scripts:

- The purge stored procedures should not be created unless the customer is going to purge data.
- Create only those stored procedures that will be used.
- Create a separate database login that will execute the purge routine.
- Assign minimal database access privileges to this user.
- Create a public synonym to provide another layer of transparency.
- Create a role that contains the privileges to execute the stored procedure.
- Assign this role to the purge user.

Note: The role with the execute procedure privilege does not need specific access to the underlying tables. In other words, when a user executes another user's procedure, the procedure is executed using the privileges of the procedure owner, not the invoker. Thus the invoker does not have direct delete privileges on the tables contained within the stored procedure.

B

Appendix: Changing and Configuring a New Base Currency

Changing Currency

In order to switch to another base and alternate currency you'll have to perform the following steps:

1. Set the base currency flag in the primary currency of the currency table. For example, if EUR is the base currency:

   ```
   update co_cny set FL_CNY_BASE='1' where DE_CNY='EUR'
   ```

2. Remove the base currency flag from any other currencies in that table. For example:

   ```
   update co_cny set FL_CNY_BASE = '0' where DE_CNY <> 'EUR'
   ```

3. Enforce ordering so that the primary currency is first and the alternate currency is second for the AI_CNY_PRI column in the currency table. Other rows should be ordered, but the specific order isn't important. For example if EUR is base currency and GBP is the alternate:

   ```
   update co_cny set AI_CNY_PRI=0 where DE_CNY='EUR'
   update co_cny set AI_CNY_PRI=1 where DE_CNY='GBP'
   update co_cny set AI_CNY_PRI=2 where DE_CNY='USD'
   update co_cny set AI_CNY_PRI=3 where DE_CNY='CAD'
   update co_cny set AI_CNY_PRI=4 where DE_CNY='MXN'
   update co_cny set AI_CNY_PRI=5 where DE_CNY='JPY'
   ```

4. Add store safe tenders supported for the new base/alternate currency. For example, if EUR is the new base currency, add money order tender support for EUR:

   ```
   insert into le_tnd_str_sf
   (ID_RPSTY_TND, TY_TND, TY_SB_TND, LU_CNY_ISSG_CY, TS_CRT_RCRD, TS_MDF_RCRD, ID_
   CNY_ICD )
   VALUES ('1','MNYO', ' ', 'EU', CURRENT_TIMESTAMP, CURRENT_TIMESTAMP, 5);
   ```

   Remove store safe tenders no longer support for the old base/alternate currency. For example, if USD if the old base currency, remove money order tender support for USD:

   ```
   delete from le_tnd_str_sf where LU_CNY_ISSG_CY = 'US' and TY_TND = 'MNYO';
   ```

5. Add exchange rate records for alternate currencies into the CO_RT_EXC table based on the new base currency. Delete all exchange rate records based on any previous base currency.

There are some application parameters that must be changed as well:

- Tender Group:
  - `CashAccepted`: For example, if EUR is base and GBP is alternate, make sure that the `CashAccepted` parameter is changed so that EUR and GBP are selected.
  - `TravelersChecksAccepted`: For EUR as base and GBP as alternate, the values for the `TravelersChecksAccepted` parameter should be EURCHK and GBPCHK.
  - `ChecksAccepted`: For EUR as base and GBP as alternate, the values for the `ChecksAccepted` parameter should be EURCHK and GBPCHK.
  - `GiftCertificatesAccepted`
  - `StoreCreditsAccepted`
- Reconciliation Group:
  - `TendersToCountAtTillReconcile`: For EUR as base and GBP as alternate, the values for the `TendersToCountAtTillReconcile` parameter should be:
    - Cash
    - Check
    - ECheck
    - Credit
    - Debit
    - TravelCheck
    - GiftCert
    - Coupon
    - GiftCard
    - StoreCredit
    - MallCert
    - PurchaseOrder
    - MoneyOrder
    - GBPCash
    - GBPTravelCheck
    - GBPCheck
    - GBPGiftCert
    - GBPStoreCredit

Configuring a New Base Currency

Throughout this section, "Krona" is used as the example new base currency that is being configured. The Krona currency code is SEK, and the issuing country code is SE.

Currency SQL Configuration

Currency Table CO_CNY

A new record describing the new currency information such as its currency code, issuing country code and so forth, must be inserted into this table.

In the base currency flag column **FL_CNY_BASE**, the new currency must be set to **1** indicating that it is the base. The flag for other currencies must be set to **0**, indicating that they are alternate currencies.

> **Note:** Point-of-Service supports base-plus-one alternate currency. The priority column AI_CNY_PRI must be set to 0 for the new base currency. It must be set to 1 for the supported alternate currency. For other alternate currencies, they must be ordered and greater than 1, but the specific order isn't important.

For example:

Example B–1 Add Krona as Base to Currency Table CO_CNY

```
INSERT INTO CO_CNY
(ID_CNY_ICD, LU_CNY_ISSG_CY, CD_CNY_ISO, DE_CNY, DE_CNY_ISSG_NAT, FL_CNY_BASE, QU_
CNY_SCLE, AI_CNY_PRI)
VALUES (7,'SE', 'SEK', 'SEK', 'Sweden', '1', 2, 0);

UPDATE CO_CNY
SET FL_CNY_BASE = '0'
WHERE CD_CNY_ISO <> 'SEK';

UPDATE CO_CNY
SET AI_CNY_PRI = AI_CNY_PRI + 1
WHERE CD_CNY_ISO <> 'SEK';
```

Currency Denomination Table CO_CNY_DNM

Denominations for the new base currency must be added to the CO_CNY_DNM table. For example:

Example B–2 Add Krona Denominations to Denomination Table CO_CNY_DNM

```
INSERT INTO CO_CNY_DNM
(ID_CNY_ICD, ID_CNY_DNM, NM_DNM, VL_DNM, CD_DNM_DPLY_PRI)
VALUES (7, 1, 'SE_50Ores', '0.50', 1);

INSERT INTO CO_CNY_DNM
(ID_CNY_ICD, ID_CNY_DNM, NM_DNM, VL_DNM, CD_DNM_DPLY_PRI)
VALUES (7, 2, 'SE_1Kronas', '1.00', 2);

INSERT INTO CO_CNY_DNM
(ID_CNY_ICD, ID_CNY_DNM, NM_DNM, VL_DNM, CD_DNM_DPLY_PRI)
VALUES (7, 3, 'SE_5Kronas', '5.00', 3);

INSERT INTO CO_CNY_DNM
(ID_CNY_ICD, ID_CNY_DNM, NM_DNM, VL_DNM, CD_DNM_DPLY_PRI)
VALUES (7, 4, 'SE_10Kronas', '10.00', 4);

INSERT INTO CO_CNY_DNM
```

```
(ID_CNY_ICD, ID_CNY_DNM, NM_DNM, VL_DNM, CD_DNM_DPLY_PRI)
VALUES (7, 5, 'SE_20Kronas', '20.00', 5);

INSERT INTO CO_CNY_DNM
(ID_CNY_ICD, ID_CNY_DNM, NM_DNM, VL_DNM, CD_DNM_DPLY_PRI)
VALUES (7, 6, 'SE_50Kronas', '50.00', 6);

INSERT INTO CO_CNY_DNM
(ID_CNY_ICD, ID_CNY_DNM, NM_DNM, VL_DNM, CD_DNM_DPLY_PRI)
VALUES (7, 7, 'SE_100Kronas', '100.00', 7);

INSERT INTO CO_CNY_DNM
(ID_CNY_ICD, ID_CNY_DNM, NM_DNM, VL_DNM, CD_DNM_DPLY_PRI)
VALUES (7, 8, 'SE_1000Kronas', '1000.00', 8);
```

Exchange Rate Table CO_RT_EXC

Add exchange rate records for alternate currencies into the CO_RT_EXC table based on the new base currency. Delete all exchange rate records based on any previous base currency. For example:

Example B–3 Add Alternate Currency Exchange Rates to Krona

```
-- Delete all the existing records
Delete from CO_RT_EXC;

INSERT INTO CO_RT_EXC
(LL_CNY_EXC, DC_RT_EXC_EF, DC_RT_EXC_EP, ID_CNY_ICD, MO_RT_TO_BUY, MO_RT_TO_SL,
MO_FE_SV_EXC)
VALUES(0.00, TO_DATE('1990-01-01', 'YYYY-MM-DD'), TO_DATE('2099-12-31',
'YYYY-MM-DD'), 1, 6.3337, 6.3362, 0.00);

INSERT INTO CO_RT_EXC
(LL_CNY_EXC, DC_RT_EXC_EF, DC_RT_EXC_EP, ID_CNY_ICD, MO_RT_TO_BUY, MO_RT_TO_SL,
MO_FE_SV_EXC)
VALUES(0.00, TO_DATE('1990-01-01', 'YYYY-MM-DD'), TO_DATE('2099-12-31',
'YYYY-MM-DD'), 2, 6.2849, 6.2898, 0.00);

INSERT INTO CO_RT_EXC
(LL_CNY_EXC, DC_RT_EXC_EF, DC_RT_EXC_EP, ID_CNY_ICD, MO_RT_TO_BUY, MO_RT_TO_SL,
MO_FE_SV_EXC)
VALUES(0.00, TO_DATE('1990-01-01', 'YYYY-MM-DD'), TO_DATE('2099-12-31',
'YYYY-MM-DD'), 3, 0.5799, 0.5816, 0.00);

INSERT INTO CO_RT_EXC
(LL_CNY_EXC, DC_RT_EXC_EF, DC_RT_EXC_EP, ID_CNY_ICD, MO_RT_TO_BUY, MO_RT_TO_SL,
MO_FE_SV_EXC)
VALUES(0.00, TO_DATE('1990-01-01', 'YYYY-MM-DD'), TO_DATE('2099-12-31',
'YYYY-MM-DD'), 4, 12.434, 12.441, 0.00);

INSERT INTO CO_RT_EXC
(LL_CNY_EXC, DC_RT_EXC_EF, DC_RT_EXC_EP, ID_CNY_ICD, MO_RT_TO_BUY, MO_RT_TO_SL,
MO_FE_SV_EXC)
VALUES(0.00, TO_DATE('1990-01-01', 'YYYY-MM-DD'), TO_DATE('2099-12-31',
'YYYY-MM-DD'), 5, 9.3739, 9.3796, 0.00);

INSERT INTO CO_RT_EXC
(LL_CNY_EXC, DC_RT_EXC_EF, DC_RT_EXC_EP, ID_CNY_ICD, MO_RT_TO_BUY, MO_RT_TO_SL,
MO_FE_SV_EXC)
VALUES(0.00, TO_DATE('1990-01-01', 'YYYY-MM-DD'), TO_DATE('2099-12-31',
'YYYY-MM-DD'), 6, 0.05782, 0.05786, 0.00);

INSERT INTO CO_RT_EXC
(LL_CNY_EXC, DC_RT_EXC_EF, DC_RT_EXC_EP, ID_CNY_ICD, MO_RT_TO_BUY, MO_RT_TO_SL,
MO_FE_SV_EXC)
VALUES(0.00, TO_DATE('1990-01-01', 'YYYY-MM-DD'), TO_DATE('2099-12-31',
'YYYY-MM-DD'), 7, 1.0, 1.0, 0.00);
```

Store Safe Tender Table LE_TND_STR_SF

Add the store safe tenders supported for the new base currency. For example:

Example B–4 Add Store Safe Tenders for Krona

```
INSERT INTO LE_TND_STR_SF
    ( ID_RPSTY_TND, TY_TND, TY_SB_TND, LU_CNY_ISSG_CY, TS_CRT_RCRD, TS_MDF_RCRD,
ID_CNY_ICD )
    VALUES('1','CASH', ' ', 'SE', CURRENT_TIMESTAMP, CURRENT_TIMESTAMP, 7);
INSERT INTO LE_TND_STR_SF
    ( ID_RPSTY_TND, TY_TND, TY_SB_TND, LU_CNY_ISSG_CY, TS_CRT_RCRD, TS_MDF_RCRD,
ID_CNY_ICD )
    VALUES('1','CHCK', ' ', 'SE', CURRENT_TIMESTAMP, CURRENT_TIMESTAMP, 7);
INSERT INTO LE_TND_STR_SF
    ( ID_RPSTY_TND, TY_TND, TY_SB_TND, LU_CNY_ISSG_CY, TS_CRT_RCRD, TS_MDF_RCRD,
ID_CNY_ICD )
    VALUES('1','TRAV', ' ', 'SE', CURRENT_TIMESTAMP, CURRENT_TIMESTAMP, 7);

-- MoneyOrderSafeTender

INSERT INTO LE_TND_STR_SF
(ID_RPSTY_TND, TY_TND, TY_SB_TND, LU_CNY_ISSG_CY, TS_CRT_RCRD, TS_MDF_RCRD, ID_
CNY_ICD )
VALUES ('1','MNYO', ' ', 'SE', CURRENT_TIMESTAMP, CURRENT_TIMESTAMP, 7);
```

Parameter Configuration

The following tender parameters must be enhanced to include the new base currency:

- StoreCreditsAccepted
- ChecksAccepted
- CashAccepted
- GiftCertificatesAccepted
- TravelersChecksAccepted

The reconciliation parameter **TendersToCountAtTillReconcile** parameter must include all the tenders to count for both base and alternate currencies during till reconcilation. For example:

Example B–5 Parameters to Support Krona as the Base and USD as the Alternate Currency

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE SOURCE PUBLIC "SOURCE"
"classpath://com/extendyourstore/foundation/tour/dtd/paramsourcescript.dtd">
<SOURCE name="register">
<GROUP hidden="N" name="Tender">
<PARAMETER final="N" hidden="N" name="StoreCreditsAccepted" type="LIST">
```

```
<VALIDATOR class="EnumeratedListValidator"
package="com.extendyourstore.foundation.manager.parameter">
<PROPERTY propname="member" propvalue="None"/>
<PROPERTY propname="member" propvalue="SEK"/>
<PROPERTY propname="member" propvalue="USD"/>
<PROPERTY propname="member" propvalue="CAD"/>
<PROPERTY propname="member" propvalue="MXN"/>
<PROPERTY propname="member" propvalue="EUR"/>
<PROPERTY propname="member" propvalue="JPY"/>
<PROPERTY propname="member" propvalue="GBP"/>
</VALIDATOR>
<VALUE value="SEK"/>
<VALUE value="USD"/>
<VALUE value="CAD"/>
<VALUE value="MXN"/>
<VALUE value="EUR"/>
<VALUE value="JPY"/>
<VALUE value="GBP"/>
</PARAMETER>
<PARAMETER final="N" hidden="N" name="ChecksAccepted" type="LIST">
<VALIDATOR class="EnumeratedListValidator"
package="com.extendyourstore.foundation.manager.parameter">
<PROPERTY propname="member" propvalue="None"/>
<PROPERTY propname="member" propvalue="SEKCHK"/>
<PROPERTY propname="member" propvalue="USDCHK"/>
<PROPERTY propname="member" propvalue="CADCHK"/>
<PROPERTY propname="member" propvalue="EURCHK"/>
<PROPERTY propname="member" propvalue="GBPCHK"/>
</VALIDATOR>
<VALUE value="SEKCHK"/>
<VALUE value="USDCHK"/>
</PARAMETER>
<PARAMETER final="N" hidden="N" name="CashAccepted" type="LIST">
<VALIDATOR class="EnumeratedListValidator"
package="com.extendyourstore.foundation.manager.parameter">
<PROPERTY propname="member" propvalue="None"/>
<PROPERTY propname="member" propvalue="SEK"/>
<PROPERTY propname="member" propvalue="USD"/>
<PROPERTY propname="member" propvalue="CAD"/>
<PROPERTY propname="member" propvalue="GBP"/>
<PROPERTY propname="member" propvalue="EUR"/>
</VALIDATOR>
<VALUE value="SEK"/>
<VALUE value="USD"/>
</PARAMETER>
<PARAMETER final="N" hidden="N" name="GiftCertificatesAccepted" type="LIST">
<VALIDATOR class="EnumeratedListValidator"
package="com.extendyourstore.foundation.manager.parameter">
<PROPERTY propname="member" propvalue="None"/>
<PROPERTY propname="member" propvalue="SEK"/>
<PROPERTY propname="member" propvalue="USD"/>
<PROPERTY propname="member" propvalue="CAD"/>
<PROPERTY propname="member" propvalue="MXN"/>
<PROPERTY propname="member" propvalue="EUR"/>
<PROPERTY propname="member" propvalue="JPY"/>
<PROPERTY propname="member" propvalue="GBP"/>
</VALIDATOR>
<VALUE value="SEK"/>
<VALUE value="USD"/>
<VALUE value="CAD"/>
```

```
<VALUE value="MXN"/>
<VALUE value="EUR"/>
<VALUE value="JPY"/>
<VALUE value="GBP"/>
</PARAMETER>
<PARAMETER final="N" hidden="N" name="TravelersChecksAccepted" type="LIST">
<VALIDATOR class="EnumeratedListValidator"
package="com.extendyourstore.foundation.manager.parameter">
<PROPERTY propname="member" propvalue="None"/>
<PROPERTY propname="member" propvalue="SEKCHK"/>
<PROPERTY propname="member" propvalue="USDCHK"/>
<PROPERTY propname="member" propvalue="CADCHK"/>
<PROPERTY propname="member" propvalue="GBPCHK"/>
<PROPERTY propname="member" propvalue="EURCHK"/>
</VALIDATOR>
<VALUE value="SEKCHK"/>
<VALUE value="USDCHK"/>
</PARAMETER>
</GROUP>
<GROUP hidden="N" name="Reconciliation">
<PARAMETER final="N" hidden="N" name="TendersToCountAtTillReconcile" type="LIST">
<VALIDATOR class="EnumeratedListValidator"
package="com.extendyourstore.foundation.manager.parameter">
<PROPERTY propname="member" propvalue="Cash"/>
<PROPERTY propname="member" propvalue="Check"/>
<PROPERTY propname="member" propvalue="ECheck"/>
<PROPERTY propname="member" propvalue="Credit"/>
<PROPERTY propname="member" propvalue="Debit"/>
<PROPERTY propname="member" propvalue="TravelCheck"/>
<PROPERTY propname="member" propvalue="GiftCert"/>
<PROPERTY propname="member" propvalue="Coupon"/>
<PROPERTY propname="member" propvalue="GiftCard"/>
<PROPERTY propname="member" propvalue="StoreCredit"/>
<PROPERTY propname="member" propvalue="MallCert"/>
<PROPERTY propname="member" propvalue="PurchaseOrder"/>
<PROPERTY propname="member" propvalue="MoneyOrder"/>
<PROPERTY propname="member" propvalue="USDCash"/>
<PROPERTY propname="member" propvalue="USDTravelCheck"/>
<PROPERTY propname="member" propvalue="USDCheck"/>
<PROPERTY propname="member" propvalue="USDGiftCert"/>
<PROPERTY propname="member" propvalue="USDStoreCredit"/>
<PROPERTY propname="member" propvalue="CADCash"/>
<PROPERTY propname="member" propvalue="CADTravelCheck"/>
<PROPERTY propname="member" propvalue="CADCheck"/>
<PROPERTY propname="member" propvalue="CADGiftCert"/>
<PROPERTY propname="member" propvalue="CADStoreCredit"/>
<PROPERTY propname="member" propvalue="MXNGiftCert"/>
<PROPERTY propname="member" propvalue="MXNStoreCredit"/>
<PROPERTY propname="member" propvalue="GBPGiftCert"/>
<PROPERTY propname="member" propvalue="GBPStoreCredit"/>
<PROPERTY propname="member" propvalue="GBPCash"/>
<PROPERTY propname="member" propvalue="GBPTravelCheck"/>
<PROPERTY propname="member" propvalue="GBPCheck"/>
<PROPERTY propname="member" propvalue="EURCash"/>
<PROPERTY propname="member" propvalue="EURTravelCheck"/>
<PROPERTY propname="member" propvalue="EURCheck"/>
<PROPERTY propname="member" propvalue="EURGiftCert"/>
<PROPERTY propname="member" propvalue="EURStoreCredit"/>
<PROPERTY propname="member" propvalue="JPYGiftCert"/>
<PROPERTY propname="member" propvalue="JPYStoreCredit"/>
```

```
</VALIDATOR>
<VALUE value="Cash"/>
<VALUE value="Check"/>
<VALUE value="ECheck"/>
<VALUE value="Credit"/>
<VALUE value="Debit"/>
<VALUE value="TravelCheck"/>
<VALUE value="GiftCert"/>
<VALUE value="Coupon"/>
<VALUE value="GiftCard"/>
<VALUE value="StoreCredit"/>
<VALUE value="MallCert"/>
<VALUE value="PurchaseOrder"/>
<VALUE value="MoneyOrder"/>
<VALUE value="USDCash"/>
<VALUE value="USDTravelCheck"/>
<VALUE value="USDCheck"/>
<VALUE value="USDGiftCert"/>
<VALUE value="USDStoreCredit"/>
<VALUE value="CADCash"/>
<VALUE value="CADTravelCheck"/>
<VALUE value="CADCheck"/>
<VALUE value="CADGiftCert"/>
<VALUE value="CADStoreCredit"/>
<VALUE value="MXNGiftCert"/>
<VALUE value="MXNStoreCredit"/>
<VALUE value="GBPGiftCert"/>
<VALUE value="GBPStoreCredit"/>
<VALUE value="EURGiftCert"/>
<VALUE value="EURStoreCredit"/>
<VALUE value="JPYGiftCert"/>
<VALUE value="JPYStoreCredit"/>
</PARAMETER>
</GROUP>
</SOURCE>
```

Resource Bundle Configuration

Resource bundle keys describing the new currency's denominations must be added to Back Office resource bundle `currency.properties`. For example:

Example B–6 New currency.properties Resource Bundle Keys

```
currency.SE_50Ores=50 Ores
currency.SE_1Kronas=1 Kronas
currency.SE_5Kronas=5 Kronas
currency.SE_10Kronas=10 Kronas
currency.SE_20Kronas=20 Kronas
currency.SE_50Kronas=50 Kronas
currency.SE_100Kronas=100 Kronas
currency.SE_1000Kronas=1000 Kronas
```

C Appendix: Audit Logging

The audit log retains events that are logged to the file system. Audit Logs include access, search, view (generate), print and export for the following functional areas in Back Office.

- Role Security (Role Audit Log Events)
- Password Policy (Password Audit Log Events)
- Users (Login, Logout, Lockout Audit Log Events)
- Employees (Employee Audit Log Events)
- Store Open, Store Close, Register Open, Register Close and Bank Deposit (Daily Operations Audit Log Events)
- Till Open, Till Reconcile and Till Close (Till Audit Log Events)
- Parameter Log Events

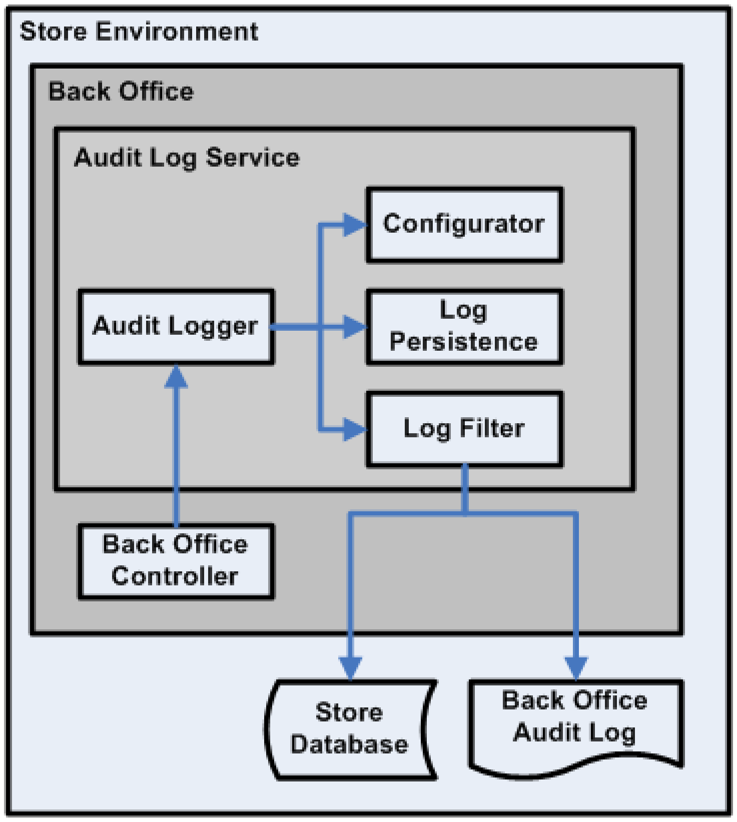
Each event has a specific set of components that must be present in the Audit Log. Each event is required to have an event name, event status, system date and system time in which the event was completed. The status of an event can either be Success or Failure. If an event was executed without interruption and the data of the event is saved to persistent storage, the events status is **Success**. If a database exception occurs after the operator or system has finished the event, the events status is **Failure**. If any exception occurs before the activity is saved or if the operator selects to leave the application, no event is logged.

Back Office events must include the store number of the events location as well as the user ID of the employee performing the event. Back Office daily operations and till options events not related to a generated report also include the business date. This enables retailers to identify events that could have occurred on a different business date and system date.

The Audit Log is implemented using a log4j logging infrastructure.

The following is a Back Office common configuration for the Audit Logging subsystem:

Figure C–1 Audit Log in Back Office



Configuring the Audit Log

Note: The Audit Log is a PABP requirement, and must not be disabled in any capacity.

For more information on PABP, see the *Oracle Retail Strategic Store Solutions Security Implementation Guide*. The guide is available on Metalink:

Metalink Note: 567438.1

You can configure the Audit Log using configuration files. To update the logging infrastructure, update the Spring `ServiceContext.xml` file to point the various infrastructure bean IDs to any alternate implementation classes you want to provide.

```
Bean ID: service_AuditLogger
Class: com._360commerce.commerceservices.audit.AuditLoggerService
```

Because the Audit Log is using Log4J as the underlying logging mechanism, you can also control the logging layout, location, and content by updating the `log4j.properties` file.

- All log events will be logged at the INFO level, so to disable logging entirely, change the log level to WARN or above for the event package path.

- Additionally, each logging event is represented in the `log4j.properties` file through the event's package path, so to filter a specific event, just update that event's level to WARN or above.
- As with all Log4J deployments, updating the layout of the log events or their location is a matter of setting the layout in the configuration file and updating the appender to point to a different file name. Another option is to use an entirely different appender to write to a database or even a JMS queue.

The following is an example of settings that might be used in a log4j.properties file:

Example C–1 Audit Log Configuration Changes in the log4j.properties File

```
# Audit Logger Settings
log4j.appender.A=org.apache.log4j.RollingFileAppender
log4j.appender.A.File=@deploy.audit.log.file@
log4j.appender.A.MaxFileSize=100000KB
log4j.appender.A.layout=org.apache.log4j.PatternLayout
log4j.appender.A.layout.ConversionPattern=%m%n

log4j.logger.com._360commerce.commerceservices.audit.event=false
log4j.additivity.com._360commerce.commerceservices.audit.event=false

log4j.logger.com._360commerce.commerceservices.audit.event.ENTER_BUSINESS_
DATE=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.START_OF_DAY=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.END_OF_DAY=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.BANK_DEPOSIT=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.BANK_DEPOSIT_REPORT_
VIEWED=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.BANK_DEPOSIT_REPORT_
EXPORTED=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.REGISTER_OPEN=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.REGISTER_CLOSE=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.TILL_RECONCILE=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.TILL_OPEN=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.COUNT_FLOAT_AT_
RECONCILE=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.ADD_EMPLOYEE=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.MODIFY_EMPLOYEE_
INFORMATION=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.ADD_TEMPORARY_
EMPLOYEE=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.MODIFY_TEMPORARY_
EMPLOYEE_INFORMATION=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.RESET_EMPLOYEE_
PASSWORD=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.RESET_TEMPORARY_
EMPLOYEE_PASSWORD=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.CHANGE_PASSWORD=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.USER_LOGOUT=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.USER_LOGIN=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.USER_LOCK_OUT=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.ADD_ROLE=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.EDIT_ROLE=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.REMOVE_ROLE=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.MODIFY_APPLICATION_
PARAMETER=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.MODIFY_PARAMETER_IN_
LIST=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.ADD_PARAMETER_LIST_FOR_
```

```
DISTRIBUTION=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.REMOVE_PARAMETER_
LIST=INFO,A
log4j.logger.com._360commerce.commerceservices.audit.event.DISTRIBUTE_PARAMETER_
LIST=INFO,A
```

Daily Operations Audit Log Events

Enter Business Date

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the Enter Business Date system setting = INFO.

- Event data collection starts when the operator enters a business date.
- Event data collection ends when the operator selects **Next**.
- There is no failure condition to this event.

Table C–1 **Enter Business Date Event Components**

Event Components	Notes
Event Name	Enter Business Date
Event Status	Success
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
Business Date	Entered Business date.
System Time	Time of the event.
Store ID	Store number the event transpired at.
User ID	User ID performing the event.

Start of Day

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the Start of Day system setting = INFO.

- Event data collection starts when the operator selects to execute Start of Day functionality.
- Event data collection ends when the system displays that the store is opened.
- The format of this event is dependent on the Count Operating Fund at Start of Day parameter setting.
- Failure can happen only when there is some technical error.

Table C–2 **Start of Day Event Components**

Event Components	Notes
Event Name	Start of Day
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
Business Date	Business date of the event.
System Time	Time of the event.
Store ID	Store number the event transpired at.
User ID	User ID performing the event.
Operating Fund Amount	■ Entered cash amount for Count Operating Fund at Start of Day = Summary. ■ Total of entered cash amount for Count Operating Fund at Start of Day = Detail. ■ Equal to the Operating Fund Expected Amount when Count Operating Fund at Start of Day = No.
Pennies	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day = Detail.
Nickels	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day = Detail.
Dimes	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day = Detail.
Quarters	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day = Detail.
Half-Dollars	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day = Detail.
$1 Coins	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day = Detail.
$2 Coins	Entered currency denomination amount. Only recorded if a value is entered, Count Operating Fund at Start of Day = Detail and Canadian currency is the base currency.
$1 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day = Detail.
$2 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day = Detail.
$5 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day = Detail.
$10 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day = Detail.
$20 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day = Detail.

Table C–2 **Start of Day Event Components**

Event Components	Notes
$50 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day = Detail.
$100 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at Start of Day = Detail.
Store Status	■ Open ■ Close

End of Day

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the End of Day system setting = INFO.

- Event data collection starts when the operator selects to begin end of day.
- Event data collection ends when the system assigns a transaction number.
- The format of this event is dependent on the Count Operating Fund at End of Day parameter setting.
- Event failure can happen only due to technical reasons, for example, unable to get next sequence number for transaction, transaction creation exception, EJB call exception or if the financial totals are not found in the database.

Table C–3 **End of Day Event Components**

Event Components	Notes
Event Name	End of Day
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
Business Date	Business date of the event.
System Time	Time of the event.
Store ID	Store number the event transpired at.
User ID	User ID performing the event.
Suspended Transactions	Record the number of suspended transactions. If no suspended transactions exist this data is not recorded in the audit log.
Suspended Transaction Report Viewed	■ True ■ False
Suspended Transaction Report Exported	■ True ■ False
Operating Fund Amount	■ Entered cash amount for Count Operating Fund at End of Day = Summary. ■ Total of entered cash amount for Count Operating Fund at End of Day = Detail. ■ Equal to the Operating Fund Expected Amount when Count Operating Fund at End of Day = No.

Table C–3 **End of Day Event Components**

Event Components	Notes
Pennies	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day = Detail.
Nickels	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day = Detail.
Dimes	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day = Detail.
Quarters	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day = Detail.
Half-Dollars	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day = Detail.
$1 Coins	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day = Detail.
$1 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day = Detail.
$2 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day = Detail.
$5 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day = Detail.
$10 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day = Detail.
$20 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day = Detail.
$50 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day = Detail.
$100 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Operating Fund at End of Day = Detail.
Transaction Number	Transaction number assigned by the system to the store close event.

Bank Deposit

This is a Back Office event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the Bank Deposit system setting = INFO.

- Event data collection starts when the operator selects to create a Bank Deposit.
- Event data collection ends when the system assigns a transaction number.
- The format of this event is dependent on the Count Deposit Tender parameter setting.
- Event failure can happen only due to technical reasons, e.g. unable to get next sequence number for transaction, transaction creation exception, and EJB or Database exceptions.

Table C–4 **Bank Deposit Event Components**

Event Components	Notes
Event Name	Bank Deposit
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
Business Date	Business date of the event.
System Time	Time of the event.
Store ID	Store number the event transpired at.
User ID	User ID performing the event.
Cash Total	■ Entered amount when Count Deposit Tender = Summary. ■ Total amount all denominations entered when Count Deposit Tender = Detail.
Pennies	Entered currency denomination amount. Only recorded if a value is entered and Count Deposit Tender = Detail.
Nickels	Entered currency denomination amount. Only recorded if a value is entered and Count Deposit Tender = Detail.
Dimes	Entered currency denomination amount. Only recorded if a value is entered and Count Deposit Tender = Detail.
Quarters	Entered currency denomination amount. Only recorded if a value is entered and Count Deposit Tender = Detail.
Half-Dollars	Entered currency denomination amount. Only recorded if a value is entered and Count Deposit Tender = Detail.
$1 Coins	Entered currency denomination amount. Only recorded if a value is entered and Count Deposit Tender = Detail.
$1 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Deposit Tender = Detail.
$2 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Deposit Tender = Detail.
$5 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Deposit Tender = Detail.
$10 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Deposit Tender = Detail.
$20 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Deposit Tender = Detail.
$50 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Deposit Tender = Detail.
$100 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Deposit Tender = Detail.
Deposited Check Total	■ Entered amount when Count Deposit Tender = Summary. ■ Total amount all deposited checks entered when Count Deposit Tender = Detail.
<ARG> Check	■ Check amount entered when Count Deposit Tender = Detail. ■ There is an audit log entry for each check entered. ■ <ARG> = the number of the check.

Table C–4 **Bank Deposit Event Components**

Event Components	Notes
Money Order Total	■ Entered amount when Count Deposit Tender = Summary. ■ Total amount all money orders entered when Count Deposit Tender = Detail.
<ARG> Money Order	■ Money Order amount entered when Count Deposit Tender = Detail. ■ There is an audit log entry for each money order entered. ■ <ARG> = the number of the money order.
Travelers Check Total	■ Entered amount when Count Deposit Tender = Summary. ■ Total amount all travelers checks entered when Count Deposit Tender = Detail.
<ARG> Travelers Check	■ Travelers check amount entered when Count Deposit Tender = Detail. ■ There is an audit log entry for each travelers check entered. ■ <ARG> = the number of the travelers check.
Canadian Cash Total	■ Entered amount when Count Deposit Tender = Summary. ■ Total amount all Canadian cash entered when Count Deposit Tender = Detail.
$2 Coins	Entered currency denomination amount. Only recorded if a value is entered and Count Deposit Tender = Detail.
Canadian Check Total	■ Entered amount when Count Deposit Tender = Summary. ■ Total amount all denominations entered when Count Deposit Tender = Detail.
<ARG> Canadian Check	■ Canadian check amount entered when Count Deposit Tender = Detail. ■ There is an audit log entry for each Canadian check entered. ■ <ARG> = the number of the Canadian check.
Canadian Travelers Check Total	■ Entered amount when Count Deposit Tender = Summary. ■ Total amount all denominations entered when Count Deposit Tender = Detail.
<ARG> Canadian: Travelers Check	■ Canadian travelers check amount entered when Count Deposit Tender = Detail. ■ There is an audit log entry for each Canadian travelers check entered. ■ <ARG> = the number of the Canadian travelers check.
Total	Contains the total amounts for local Cash, Check, Money Order and Travelers Checks.
Canadian Total	Contains the total amounts for Canadian Cash, Canadian Travelers Check, and Canadian Checks.
Transaction Number	Transaction number assigned by the system to the bank deposit event.

Bank Deposit Report Viewed

This is a Back Office event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the Bank Deposit Report Viewed system setting = INFO.

- Event data collection starts and ends when the system displays the Bank Deposit Report.
- Event failure can happen only due to technical reasons or database exceptions.

Table C–5 **Bank Deposit Report Viewed Event Components**

Event Components	Notes
Event Name	Bank Deposit Report Viewed
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
Business Date	Business date of the event.
System Time	Time of the event.
Store ID	Store number the event transpired at.
User ID	User ID performing the event.

Bank Deposit Report Exported

This is a Back Office event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the Bank Deposit Report Exported system setting = INFO.

- Event data collection starts when the operator selects to export the Bank Deposit Report.
- Event data collection ends when the operator selects Save.
- Event failure can happen only when there is a technical exception.

Table C–6 **Bank Deposit Report Exported Event Components**

Event Components	Notes
Event Name	Bank Deposit Report Exported
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
Business Date	Business date of the event.
System Time	Time of the event.

Table C–6 **Bank Deposit Report Exported Event Components**

Event Components	Notes
Store ID	Store number the event transpired at.
User ID	User ID performing the event.
Export Format	■ PDF ■ RTF ■ CSV ■ XML

Register Open

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the Register Open system setting = INFO.

- Event data collection starts when the operator selects to open a register.
- Event data collection ends when the system assigns a transaction number.
- If more than one register is selected to open at one time, a separate independent event is written to the audit log. Each opened register is assigned an individual transaction number.
- Event failure can happen only due to technical reasons, e.g. unable to get next sequence number for transaction, transaction creation exception, EJB call exception or if the financial totals are not found in the database.

Table C–7 **Register Open Event Components**

Event Components	Notes
Event Name	Register Open
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
Business Date	Business date of the event.
System Time	Time of the event.
Store ID	Store number the event transpired at.
User ID	User ID performing the event.
Register Number	Selected register number.
Transaction Number	Transaction number assigned by the system to the opened register.

Register Close

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the Register Close system setting = INFO.

- Event data collection starts when the operator selects to close a register.
- Event data collection ends when the system assigns a transaction number.
- Event failure can happen only due to technical reasons, e.g. unable to get next sequence number for transaction, transaction creation exception, EJB call exception or if the financial totals are not found in the database.

Table C–8 **Register Close Event Components**

Event Components	Notes
Event Name	Register Close
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
Business Date	Business date of the event.
System Time	Time of the event.
Store ID	Store number the event transpired at.
User ID	User ID performing the event.
Register Number	Selected register number.
Transaction Number	Transaction number assigned by the system to the closed register.

Employee Audit Log Events

Modify Employee Information

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the Modify Employee Information system setting = INFO.

- Event data collection starts when the operator edits an employees information.
- Event data collection ends when the operator selects Save.
- If the operator selects Save but has not modified any employee information the event name is Modify Employee Information
- Employee getting modified is not found in the Database is the only failure condition possible.

Table C–9 **Modify Employee Information Event Components**

Event Components	**Notes**
Event Name	Modify Employee Information
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
System Time	Time of the event.
Store ID	Store number the event transpired at.
User ID	User ID performing the event.
Employee ID	Employee ID of the modified employee.

Modify Temporary Employee Information

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the Modify Temporary Employee Information system setting = INFO.

- Event data collection starts when the operator edits a temporary employees information.
- Event data collection ends when the operator selects Save.
- If the operator selects Save but has not modified any temporary employee information the event name is Modify Temporary Employee Information
- Employee getting modified is not found in the Database is the only failure condition possible.

Table C–10 **Modify Temporary Employee Information Event Components**

Event Components	**Notes**
Event Name	Modify Temporary Employee Information
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
System Time	Time of the event.
Store ID	Store number the event transpired at.
User ID	User ID performing the event.
Employee ID	Employee ID of the modified temporary employee.

Add Employee

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the Add Employee system setting = INFO.

- Event data collection starts when the operator selects to add an employee.
- Event data collection ends when the operator selects Save.
- Failure Event is when the login ID provided is already in use.

Table C–11 **Add Employee Event Components**

Event Components	Notes
Event Name	Add Employee
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
System Time	Time of the event.
Store ID	Store number the event transpired at.
User ID	User ID performing the event.
Employee ID	Employee ID.
First Name	Entered first name.
Middle Name	Entered middle name.
Last Name	Entered last name.
Employee Login ID	Entered login ID.
Role Name	Selected role.
Employee Status	Selected employee status.

Add Temporary Employee

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the Add Temporary Employee system setting = INFO.

- Event data collection starts when the operator selects to add a temporary employee.
- Event data collection ends when the operator selects Save.
- Failure Event is when the login ID provided is already in use.

Table C–12 **Add Temporary Employee Event Components**

Event Components	Notes
Event Name	Add Temporary Employee
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)

Table C–12 **Add Temporary Employee Event Components**

Event Components	Notes
System Date	System date of the event.
System Time	Time of the event.
Store ID	Store number the event transpired at.
User ID	User ID performing the event.
Employee ID	Employee ID.
First Name	Entered first name.
Middle Name	Entered middle name.
Last Name	Entered last name.
Employee Login ID	Entered login ID.
Role Name	Selected role.
Store#	Entered store number.
Days Valid	Selected remaining days valid.
Employee Status	Selected employee status.

Login, Logout, Lockout Audit Log Events

User Login

This is a Back Office and Point-of-Service and Central Office event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the User Login system setting = INFO.

- Event data collection starts when the operator enters their login information.
- Event data collection ends when the operator selects to login.
- Even failure can happen only when there is a technical exception.

Table C–13 **User Login Event Components**

Event Components	Notes
Event Name	User Login
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
System Time	Time of the event.
Store ID	Store number the event transpired at. Only applicable for Back Office.
User ID	User ID is recorded.

User Lock Out

This is a Back Office and Point-of-Service and Central Office event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the User Lock Out system setting = INFO.

- Event data collection starts and ends when the user attempts to login and is locked out due to unsuccessful login attempts or an expired password.
- No failure condition.

Table C–14 **User Lock Out Event Components**

Event Components	Notes
Event Name	User Lock Out
Event Status	■ Success
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
System Time	Time of the event.
Store ID	Store number the event transpired at. Only applicable for Back Office.
User ID	User ID is recorded.
Lockout Reason	■ <ARG> consecutive unsuccessful login attempts. (<ARG> = Number of login attempts) ■ Expired Password

User Logout

This is a Back Office and Central Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the User Logout system setting = INFO.

- Event data collection starts and ends when the user selects to log out.
- No Failure Condition.

Table C–15 **User Logout Event Components**

Event Components	Notes
Event Name	User Logout
Event Status	■ Success
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
System Time	Time of the event.
Store ID	Store number the event transpired at. Applicable only for Back Office.
User ID	User ID is recorded.

Password Audit Log Events

Change Password

This is a Back Office and Central Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the Change Password system setting = INFO.

- Event data collection starts when the operator selects or is prompted to change their password.
- Event data collection ends when the operator selects to save their new password.
- Failure Condition will occur when the Employee/User for whom the password is being changed does not exist in the DB. Also the New password supplied if it does not meet the password criteria then also a Failure condition will be logged.

Table C–16 **Change Password Event Components**

Event Components	Notes
Event Name	User Change Password
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
System Time	Time of the event.
Store ID	Store number the event transpired at. Applicable only for Back Office.
User ID	User ID is recorded.

Reset Employee Password

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the Reset Employee Password system setting = INFO.

- Event data collection starts when the operator selects to reset an employees password.
- Event data collection ends when the operator selects Yes.
- Failure Condition will be logged only in case of technical failures such as Database is down.

Table C–17 **Reset Employee Password Event Components**

Event Components	Notes
Event Name	Reset Employee Password
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
System Time	Time of the event.

Table C–17 **Reset Employee Password Event Components**

Event Components	Notes
Store ID	Store number the event transpired at.
User ID	User ID performing the event.
Employee ID	Employee ID whose password was reset

Reset Temporary Employee Password

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the Reset Temporary Employee Password system setting = INFO.

- Event data collection starts when the operator selects to reset an employees password.
- Event data collection ends when the operator selects Yes.
- Failure Condition will be logged only in case of technical failures such as DB is down.

Table C–18 **Reset Temporary Employee Password Event Components**

Event Components	Notes
Event Name	Reset Temporary Employee Password
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
System Time	Time of the event.
Store ID	Store number the event transpired at.
User ID	User ID performing the event.
Employee ID	Employee ID whose password was reset

Role Audit Log Events

Edit Role

This is a Back Office and Central Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the Edit Role system setting = INFO.

- Event data collection starts when the operator edits the role.
- Event data collection ends when the operator selects Save.
- Failure Condition only due to Technical exceptions.

Table C–19 **Edit Role Event Components**

Event Components	Notes
Event Name	Edit Role
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
System Time	Time of the event.
User ID	User ID performing the event.
Role Name	Selected role name.

Remove Role

This is a Back Office and Central Office event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the Remove Role system setting = INFO.

- Event data collection starts when the operator selects a roles checkbox to remove.
- Event data collection ends when the operator confirms to remove the role(s).
- Event is not logged when the operator does not confirm the role removal.
- If multiple roles are selected they are included in the same event but written as separate entries. Each time Remove is selected (and a role has been selected) an event is written to the audit log.
- Failure Condition only due to Technical exceptions.

Table C–20 **Remove Role Event Components**

Event Components	Notes
Event Name	Remove Role
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
System Time	Time of the event.
User ID	User ID performing the event.
Role Name	Selected role name. There is an audit log entry for each selected role setting.

Add Role

This is a Back Office and Central Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the Add Role system setting = INFO.

- Event data collection starts when the operator selects Add.
- Event data collection ends when the operator selects to save the role settings for the Role.
- Failure Condition only due to Technical exceptions

Table C–21 **Add Role Event Components**

Event Components	Notes
Event Name	Add Role
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
System Time	Time of the event.
User ID	User ID performing the event.
Role Name	Entered role name.
Role Setting	Selected role setting. Includes application full name and Feature.

Till Audit Log Events

Till Open

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the Till Open system setting = INFO.

- Event data collection starts when the operator selects to open a till.
- Event data collection ends when the system assigns a transaction number.
- The format of this event is dependent on the Count Float at Open parameter setting.
- Event failure can happen only due to technical reasons, e.g. unable to get next sequence number for transaction, transaction creation exception, EJB call exception or if the financial totals are not found in the database.

Table C–22 **Till Open Event Components**

Event Components	Notes
Event Name	Till Open
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.

Table C–22 **Till Open Event Components**

Event Components	Notes
Business Date	Business date of the event.
System Time	Time of the event.
Store ID	Store number the event transpired at.
User ID	User ID performing the event.
Register ID	Complete Register ID value is recorded.
Till ID	Complete Till ID value is recorded.
Operator ID	Operator ID is user assigned to the till not the logged in user id.
Float Amount	▪ Entered amount when Count Float at Open = Summary. ▪ Total amount all denominations entered when Count Float at Open = Detail. ▪ Equal to the Float Amount when Count Float at Open = No.
Pennies	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open = Detail.
Nickels	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open = Detail.
Dimes	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open = Detail.
Quarters	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open = Detail.
Half-Dollars	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open = Detail.
$1 Coins	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open = Detail.
$1 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open = Detail.
$2 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open = Detail.
$5 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open = Detail.
$10 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open = Detail.
$20 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open = Detail.
$50 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open = Detail.
$100 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Open = Detail.
Transaction Number	Transaction number assigned to opened till

Count Float at Reconcile

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the Count Float at Reconcile system setting = INFO.

- Event data collection starts when the system checks the Count Float at Reconcile parameter.
- Event data collection ends when the count float amount has been entered or accepted.
- The format of this event is dependent on the Count Float at Reconcile parameter setting.
- Event failure can happen only due to technical reasons, e.g. unable to get next sequence number for transaction, transaction creation exception, EJB call exception or if the financial totals are not found in the database.

Table C–23 **Count Float at Reconcile Event Components**

Event Components	**Notes**
Event Name	Count Float at Reconcile
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
Business Date	Business date of the event.
System Time	Time of the event.
Store ID	Store number the event transpired at.
User ID	User ID performing the event.
Register ID	Complete Register ID value is recorded.
Till ID	Complete Till ID value is recorded.
Operator ID	Operator ID is user assigned to the till not the logged in user ID.
Float Amount	■ Entered amount when Count Float at Reconcile = Summary. ■ Total amount all denominations entered when Count Float at Reconcile = Detail. ■ Equal to the Float Amount when Count Float at Reconcile = No.
Pennies	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile = Detail.
Nickels	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile = Detail.
Dimes	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile = Detail.
Quarters	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile = Detail.
Half-Dollars	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile = Detail.
$1 Coins	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile = Detail.

Table C–23 **Count Float at Reconcile Event Components**

Event Components	Notes
$1 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile = Detail.
$2 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile = Detail.
$5 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile = Detail.
$10 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile = Detail.
$20 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile = Detail.
$50 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile = Detail.
$100 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Float at Reconcile = Detail.

Till Reconcile

This is a Back Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the Till Reconcile system setting = INFO.

- Event data collection starts when the system checks the Count Till at Reconcile parameter.
- If the Count Till at Reconcile = No, event data collection ends when the system assigns a transaction number.
- If the Count Till at Reconcile = Detail or Summary, event data ends when the system displays the Reconcile Till Count Report.
- The format of this event is dependent on the Count Till at Reconcile parameter setting and Blind Close parameter setting.
- Event failure can happen only due to technical reasons, e.g. unable to get next sequence number for transaction, transaction creation exception, EJB call exception or if the financial totals are not found in the database.

Table C–24 **Till Reconcile Event Components**

Event Components	Notes
Event Name	Till Reconcile
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
System Time	Time of the event.
Business Date	Business date of the event.
Store ID	Store number the event transpired at.
User ID	User ID performing the event.

Table C–24 **Till Reconcile Event Components**

Event Components	Notes
Register ID	Complete Register ID value is recorded.
Till ID	Complete Till ID value is recorded.
Operator ID	Operator ID is user assigned to the till not the logged in user ID.
Cash Total	■ Entered tender amount if Count Till at Reconcile = Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close = No. ■ Total of all entered currencies Count Till at Reconcile = Detail and the currency was received.
Pennies	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile = Detail.
Nickels	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile = Detail.
Dimes	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile = Detail.
Quarters	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile = Detail.
Half-Dollars	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile = Detail.
$1 Coins	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile = Detail.
$1 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile = Detail.
$2 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile = Detail.
$5 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile = Detail.
$10 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile = Detail.
$20 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile = Detail.
$50 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile = Detail.
$100 Bills	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile = Detail.
Check Total	■ Entered tender amount if Count Till at Reconcile = Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close = No. ■ Total amount all deposited checks entered when Count Till at Reconcile = Detail.
<ARG> Check	■ Check amount entered when Count Till at Reconcile = Detail. ■ There is an audit log entry for each check entered. ■ <ARG> = the number of the Check.
Credit Total	■ Entered tender amount if Count Till at Reconcile = Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close = No. ■ Total of all entered Credit when Count Till at Reconcile = Detail and the tender was received.

Table C–24 **Till Reconcile Event Components**

Event Components	Notes
<ARG> Credit	■ Credit amount entered when Count Till at Reconcile = Detail. ■ There is an audit log entry for each credit entered. ■ <ARG> = the number of the Credit.
Debit Total	■ Entered tender amount if Count Till at Reconcile = Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close = No. ■ Total of all entered Debit when Count Till at Reconcile = Detail and the tender was received.
<ARG> Debit	■ Debit amount entered when Count Till at Reconcile = Detail. ■ There is an audit log entry for each Debit entered. ■ <ARG> = the number of the Debit.
Gift Card Total	■ Entered tender amount if Count Till at Reconcile = Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close = No. ■ Total of all entered Gift Card when Count Till at Reconcile = Detail and the tender was received.
<ARG> Gift Card	■ Gift Card amount entered when Count Till at Reconcile = Detail. ■ There is an audit log entry for each Gift Card entered. ■ <ARG> = the number of the Gift Card.
Gift Certificate Total	■ Entered tender amount if Count Till at Reconcile = Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close = No. ■ Total of all entered Gift Certificate when Count Till at Reconcile = Detail and the tender was received.
<ARG> Gift Certificate	Gift Certificate amount entered when Count Till at Reconcile = Detail. There is an audit log entry for each Gift Certificate entered.
Travelers Check Total	■ Entered tender amount if Count Till at Reconcile = Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close = No. ■ Total of all entered Travelers Check when Count Till at Reconcile = Detail and the tender was received.
<ARG> Travelers Check	■ Travelers Check amount entered when Count Till at Reconcile = Detail. ■ There is an audit log entry for each Travelers Check entered. ■ <ARG> = the number of the Travelers Check.
Coupon Total	■ Entered tender amount if Count Till at Reconcile = Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close = No. ■ Total of all entered Coupon when Count Till at Reconcile = Detail and the tender was received.
<ARG> Coupon	■ Coupon amount entered when Count Till at Reconcile = Detail. ■ There is an audit log entry for each Coupon entered. ■ <ARG> = the number of the Coupon.
Store Credit Total	■ Entered tender amount if Count Till at Reconcile = Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close = No. ■ Total of all entered Store Credit when Count Till at Reconcile = Detail and the tender was received.

Table C–24 Till Reconcile Event Components

Event Components	Notes
<ARG> Store Credit	■ Store Credit amount entered when Count Till at Reconcile = Detail. ■ There is an audit log entry for each Store Credit entered. ■ <ARG> = the number of the Store Credit.
Mall Certificate Total	■ Entered tender amount if Count Till at Reconcile = Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close = No. ■ Total of all entered Mall Certificate when Count Till at Reconcile = Detail and the tender was received.
<ARG> Mall Certificate	■ Mall Certificate amount entered when Count Till at Reconcile = Detail. ■ There is an audit log entry for each Mall Certificate entered. ■ <ARG> = the number of the Mall Certificate.
Purchase Order Total	■ Entered tender amount if Count Till at Reconcile = Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close = No. ■ Total of all entered Purchase Order when Count Till at Reconcile = Detail and the tender was received.
<ARG> Purchase Order	■ Purchase Order amount entered when Count Till at Reconcile = Detail. ■ There is an audit log entry for each Purchase Order entered. ■ <ARG> = the number of the Purchase Order.
E-Check Total	■ Entered tender amount if Count Till at Reconcile = Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close = No. ■ Total of all entered E-Check when Count Till at Reconcile = Detail and the tender was received.
<ARG> E-Check	■ E-Check amount entered when Count Till at Reconcile = Detail. ■ There is an audit log entry for each E-Check entered. ■ <ARG> = the number of the E-check.
Canadian Cash Total	■ Entered tender amount if Count Till at Reconcile = Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close = No. ■ Total of all entered currencies when Count Till at Reconcile = Detail and the currency was received.
$2 Coins	Entered currency denomination amount. Only recorded if a value is entered and Count Till at Reconcile = Detail.
Canadian Check Total	■ Entered tender amount if Count Till at Reconcile = Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close = No. ■ Total of all entered Canadian Check when Count Till at Reconcile = Detail and the tender was received.
<ARG> Canadian Check	■ Canadian Check amount entered when Count Till at Reconcile = Detail. ■ There is an audit log entry for each Canadian Check entered. ■ <ARG> = the number of the Canadian Check.
Canadian Travelers Check Total	■ Entered tender amount if Count Till at Reconcile = Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close = No. ■ Total of all entered Canadian Travelers Check when Count Till at Reconcile = Detail and the tender was received.

Table C–24 **Till Reconcile Event Components**

Event Components	Notes
<ARG> Canadian Travelers Check	▪ Canadian Travelers Check amount entered when Count Till at Reconcile = Detail. ▪ There is an audit log entry for each Canadian Travelers Check entered. ▪ <ARG> = the number of the Canadian Travelers Check.
Canadian Gift Certificate Total	▪ Entered tender amount if Count Till at Reconcile = Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close = No. ▪ Total of all entered Canadian Gift Certificate when Count Till at Reconcile = Detail and the tender was received.
<ARG> Canadian Gift Certificate	▪ Canadian Gift Certificate amount entered when Count Till at Reconcile = Detail. ▪ There is an audit log entry for each Canadian Gift Certificate entered. ▪ <ARG> = the number of the Canadian Gift Certificate.
Canadian Store Credit Total	▪ Entered tender amount if Count Till at Reconcile = Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close = No. ▪ Total of all entered Canadian Store Credit when Count Till at Reconcile = Detail and the tender was received.
<ARG> Canadian Store Credit	▪ Canadian Store Credit amount entered when Count Till at Reconcile = Detail. ▪ There is an audit log entry for each Canadian Store Credit entered. ▪ <ARG> = the number of the Canadian Store Credit.
Mexican Gift Certificate Total	▪ Entered tender amount if Count Till at Reconcile = Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close = No. ▪ Total of all entered Mexican Gift Certificate when Count Till at Reconcile = Detail and the tender was received.
<ARG> Mexican Gift Certificate	▪ Mexican Gift Certificate amount entered when Count Till at Reconcile = Detail. ▪ There is an audit log entry for each Mexican Gift Certificate entered. ▪ <ARG> = the number of the Mexican Gift Certificate.
Mexican Store Credit Total	▪ Entered tender amount if Count Till at Reconcile = Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close = No. ▪ Total of all entered Mexican Store Credit when Count Till at Reconcile = Detail and the tender was received.
<ARG> Mexican Store Credit	▪ Mexican Store Credit amount entered when Count Till at Reconcile = Detail. ▪ There is an audit log entry for each Mexican Store Credit entered. ▪ <ARG> = the number of the Mexican Store Credit.
UK Gift Certificate Total	▪ Entered tender amount if Count Till at Reconcile = Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close = No. ▪ Total of all entered UK Gift Certificate when Count Till at Reconcile = Detail and the tender was received.

Table C–24 **Till Reconcile Event Components**

Event Components	Notes
<ARG> UK Gift Certificate	▪ UK Gift Certificate amount entered when Count Till at Reconcile = Detail. ▪ There is an audit log entry for each UK Gift Certificate entered. ▪ <ARG> = the number of the UK Gift Certificate.
UK Store Credit Total	▪ Entered tender amount if Count Till at Reconcile = Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close = No. ▪ Total of all entered UK Store Credit when Count Till at Reconcile = Detail and the tender was received.
<ARG> UK Store Credit	▪ UK Store Credit amount entered when Count Till at Reconcile = Detail. ▪ There is an audit log entry for each UK Store Credit entered. ▪ <ARG> = the number of the UK Store Credit.
European Gift Certificate Total	▪ Entered tender amount if Count Till at Reconcile = Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close = No. ▪ Total of all entered European Gift Certificate when Count Till at Reconcile = Detail and the tender was received.
<ARG> European Gift Certificate	▪ European Gift Certificate amount entered when Count Till at Reconcile = Detail. ▪ There is an audit log entry for each European Gift Certificate entered. ▪ <ARG> = the number of the European Gift Certificate.
European Store Credit Total	▪ Entered tender amount if Count Till at Reconcile = Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close = No. ▪ Total of all entered European Gift Certificate when Count Till at Reconcile = Detail and the tender was received.
<ARG> European Store Credit	▪ European Store Credit amount entered when Count Till at Reconcile = Detail. ▪ There is an audit log entry for each European Store Credit entered. ▪ <ARG> = the number of the European Store Credit.
Japanese Gift Certificate Total	▪ Entered tender amount if Count Till at Reconcile = Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close = No. ▪ Total of all entered Japanese Gift Certificate when Count Till at Reconcile = Detail and the tender was received.
<ARG> Japanese Gift Certificate	▪ Japanese Gift Certificate amount entered when Count Till at Reconcile = Detail. ▪ There is an audit log entry for each Japanese Gift Certificate entered. ▪ <ARG> = the number of the Japanese Gift Certificate.
Japanese Store Credit Total	▪ Entered tender amount if Count Till at Reconcile = Summary. Only recorded if this tender was received and this tender is included in the Tenders To Count at Till Reconcile and if Blind Close = No. ▪ Total of all entered Japanese Store Credit when Count Till at Reconcile = Detail and the tender was received.

Table C–24 **Till Reconcile Event Components**

Event Components	Notes
<ARG> Japanese Store Credit	■ Japanese Store Credit amount entered when Count Till at Reconcile = Detail. ■ There is an audit log entry for each Japanese Store Credit entered. ■ <ARG> = the number of the Japanese Store Credit.
Blind Close	■ True ■ False
Transaction Number	Transaction number assigned to till reconcile.

Parameter Log Events

Modify Application Parameter

This is a Back Office and Central Office and Point-of-Service event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the Modify Application Parameter system setting = INFO.

- Event data collection starts when the operator selects a parameter to modify.
- Event data collection ends when the operator selects to save.

Table C–25 **Modify Application Parameter Event Components**

Event Components	Notes
Event Name	Modify Application Parameter
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
System Time	Time of the event.
Store ID	Store number the event transpired at. Only applicable for Back Office.
User ID	User ID performing the event.
Parameter Group	Parameter Group.
Parameter Name	Name of the Parameter.

Modify Parameter in List

This is a Back Office and Central Office event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the Modify Parameter in List system setting = INFO.

- Event data collection starts when the operator selects a parameter to modify.
- Event data collection ends when the operator selects to save.
- Failure due to technical reasons such as database is down.

Table C–26 **Modify Parameter in List Event Components**

Event Components	Notes
Event Name	Modify Parameter in List
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
System Time	Time of the event.
Store ID	Store number the event transpired at. Only applicable for Back Office.
User ID	User ID performing the event.
Parameter Group	Parameter Group.
Parameter Name	Name of the Parameter.
Parameter List Name	Name of the Parameter list modified. Applicable only when the modification is done to the parameter by selecting the parameter list name.

Add Parameter List For Distribution

This is a Back Office and Central Office event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the Add Parameter list for Distribution system setting = INFO.

- Event data collection starts when the operator selects to add a parameter list for distribution.
- Event data collection ends when the operator selects to “Save” or “Save and Distribute”.
- Failure due to technical reasons such as database is down.

Table C–27 **Add Parameter List For Distribution Event Components**

Event Components	Notes
Event Name	Add Parameter List For Distribution
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
System Time	Time of the event.

Table C–27 **Add Parameter List For Distribution Event Components**

Event Components	Notes
Store ID	Store number the event transpired at. Only applicable for Back Office.
User ID	User ID performing the event.
Parameter List Name	Name of the Parameter list added.

Remove Parameter List For Distribution

This is a Back Office and Central Office event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the Remove Parameter List for Distribution system setting = INFO.

- Event data collection starts when the operator selects to remove a parameter list for distribution.
- Event data collection ends when the operator confirms to remove.
- Failure due to technical reasons such as database is down.

Table C–28 **Remove Parameter List For Distribution Event Components**

Event Components	Notes
Event Name	Remove existing list for Parameter Distribution
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)
System Date	System date of the event.
System Time	Time of the event.
Store ID	Store number the event transpired at. Only applicable for Back Office.
User ID	User ID performing the event.
Parameter List Name	Name of the Parameter list removed.

Distribute Parameter List

This is a Back Office and Central Office event.

This event is written to the audit log if the Settings For Audit Logging system setting = INFO and the Distribute Parameter List system setting = INFO.

- Event data collection starts when the operator selects to “Save and Distribute” or when the system distributes the Parameter list based on the schedule.
- Event data collection ends when the system creates a task to distribute.
- Failure due to technical reasons such as database is down.

Table C–29 **Distribute Parameter List Event Components**

Event Components	Notes
Event Name	Parameter Distribution
Event Status	■ Success ■ Failure
Event Originator	Class Name and Method Name (ClassName.methodName)

Table C–29 **Distribute Parameter List Event Components**

Event Components	**Notes**
System Date	System date of the event.
System Time	Time of the event.
Store ID	Store number the event transpired at. Only applicable for Back Office.
User ID	User ID performing the event.
Parameter List Name	Name of the Parameter list distributed.
Task ID	Task ID created by the system for the distribution of Parameters.

Index